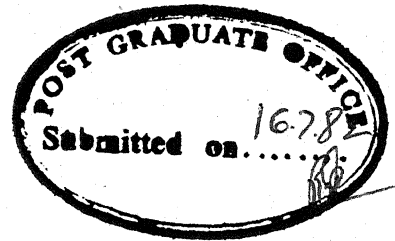# PASCAL - P IMPLEMENTATION ON TDC - 316

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

S. SRINIVAS RAGHURAM

to the
COMPUTER SCIENCE PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
JULY, 1982

CERTIFICATE

Certified that the thesis entitled "PASCAL-P IMPLEMENTATION ON TDC-316" has been carried out under my supervision by Mr. S. SRINIVAS RAGHURAM and has not been submitted elsewhere for a degree.

Kanpur

B. Srinivasan
Computer Science Programme
IIT, KANPUR

CSP-1985-M-RAG-PAS

dedicated to

the

fond memory of

my

cousin Srinivas

ACKNOWLEDGEMENTS

ABSTRACT

A single user Pascal programming station based on the Pascal-P compiler has been implemented on a 16-bit mini-computer. The original P-compiler, designed for a 60 bit CDC machine, is modified to handle variable length data types on the 16 bit machine. Run time checks for arrays and subranges have been added. Also implemented are a P-Assembler, a P-Interpreter to interpret the output of the P-Assembler, Pascal file primitives and a command processor. The addition of four relevant instructions and a transparent memory manager and reorganization of the compiler code shows superior compiler performance, based on studies conducted in a simulated environment.

CONTENTS

CHAPTER 1

INTRODUCTION

Portability is a measure of the ease with which a program can be transferred from one environment to another. Portability is important for compiler systems so that we can install the language processor on as many computer systems as possible, in a short time. The trend towards interpreter based systems is on the increase because most of the present day computer architectures are not suited for the easy implementation of advanced high level languages.

Many efforts have been made to design portable programming systems [6,15]. The Pascal-P compiler is a portable compiler for a subset of standard Pascal [12]. The Pascal-P compiler produces object code to run on a P-machine. The P-machine is an idealized stack machine. Code for this machine is termed P-code. The P-code is designed to be compact so that programs in P-code are shorter than the corresponding programs in conventional machine code. This is natural because the instruction set of the P-machine strongly reflects the constructs available in Pascal. The P-compiler is compact because the code generation for the P-machine is simple and straightforward. A P-compiler is highly portable as, it is fairly easy to implement a P-machine. PAS316, which is the subject of this report is a

single user programming system based on the P-compiler and
implemented on TDC-316.

The 'P' in P-machine and P-code stands for 'pseudo'.
The P-machine may be implemented as a physical processor or
may be emulated by an interpreter.

The P-compiler in its original form cannot be efficiently
implemented on a 16-bit minicomputer. Hence, it has been
modified to handle variable length data types. The instru-
ction set has been modified to accommodate the type infor-
mation and extended to handle local variables efficiently.
The compiler now generates code to check array and subrange
bounds.

The P-compiler produces as its object code a P-machine
assembly language program. The P-assembler assembles this
program and loads it in the P-machines code array, ready
for interpretation.

The P-code produced by the P-assembler is interpreted
by the P-interpreter. The interpreter also performs
machine dependent IO. It fetches each P-code instruction in
sequence and performs the appropriate action. The effect
of this interpretation is manifested as a change in the
state of the P-machines data segment, consisting of the Stack
and the Heap.

The environment to run the P-compiler comprises of
of about 100K bytes of memory and file-handling primitives.

The P-code contains instructions to handle text files. Pascal file IO operators have been implemented. A temporary filing system has been implemented to handle files. This pseudo-filer is interfaced with the disk through the diskio module. The basic TDC-316 computer supports direct addressing of 64 K bytes of memory. Hardware facilities for memory extension have been used and a transparent memory manager that maps 8 physical segments to 32 virtual segments each of 4 K words has been implemented.

A command processor helps the user to interact with the various subsystems of PAS316. The compilers performance, with the new instructions and the memory manager has been studied in a simulated environment on DEC 10. The segment swap rate has been observed for various test programs.

The poor performance of the P-compiler was attributed to the large number of segment faults. The reason for this is the large number of intersegment procedure calls. An attempt has been made to combine the most often used procedures into contiguous segments, by reorganizing the P-Assembly language program produced by the compiler when compiling itself. There is an appreciable performance improvement after reorganization.

The P-compiler and the various subsystems of the PAS316 system are discussed in the following chapters.

CHAPTER 2

PASCAL-P COMPILER

The Pascal-P compiler is a portable compiler for a subset of standard Pascal. The compiler is written using exactly the subset it processes and is self-compiling. Its output is an assembly language program of a hypothetical stack computer called P-machine. The stack computer is machine independent and also simple. The 'P' compiler was designed such that it can be transported from one computer to another at very little cost in terms of programming effort.

2.1 TDC-PASCAL-P

The Pascal-P compiler released by ETH, Zurich in 1973, is not suitable for implementing on mini-computers. It assumes that all the basic variable types provided in Pascal, to occupy one unit of memory. Thus an integer variable, a real variable, a character variable and a set variable are treated uniformly by the P-code load and store instructions. The initial implementation of the stack computer was on a 60 bit CDC machine. But, on a 16 bit machine like TDC-316, it is unwise to allocate equal store for all types.

Thus, it was necessary to effect modifications to the existing P-compiler at IIT Kanpur, to generate the necessary

type information with the load and store instructions of the stack computer. The P-compiler with the above modifications is implementableon TDC-316.

The P-compiler was not generating code for run-time checking of array bounds and subrange bounds. We strongly felt their necessity. They are now available and can be turned off through the switch $R-.

The P-machine instruction repertoire has been augmented with 4 new instructions. These are :

LDLt Q   ;   load variable at local address Q

STLt Q   ;   store variable at local address Q

LAL Q   ;   load local address

UJC   ;   error in case statement

't' indicates type, it takes the following values : A,B,I,R and S and denotes address, boolean, integer, real and set respectively. Local implies local to the current activation record. For loading and storing of local variables on the stack, the compiler generates the above new set of local instructions instead of LODt O,Q, STRt O,Q and LDA O,Q respectively.

The handling of the CASE statement has been modified, to indicate error in case of an undefined case label occurring at run-time.

```
VAR I : 0..5;
BEGIN
 CASE I OF
   0 : ;
   2 : ;
   4 :
 END
END.
```

```
L   3                              L   3
    ENT        L    4                  ENT        L    4
    LDO             8                  LDOI            8
    UJP        L    5                  UJP        L    5
L   7                              L   7
    UJP        L    6                  UJP        L    6
L   8                              L   8
    UJP        L    6                  UJP        L    6
L   9                              L   9
    UJP        L    6                  UJP        L    6
    L    5                         L   5
    STR    0        9                  STLI            9
I   10                             I   10
    LOD    0        9                  LDLI            9
    LDCI           0                   CHKS        0        4
    GEQI                               LDCI            0
    FJP        L    6                  SBI
    LOD    0        9                  XJP        L   10
    LDCI           4              L   10
    LEQI                               UJP        L    7
    FJP        L    6                  UJC
    LOD    0        9                  UJP        L    8
    LDCI           0                   UJC
I   20                                 UJP        L    9
    SBI                           L   6
    XJP        L   10            I   20
L   10                               RETP
    UJP        L    7            L   4=            9
    UJP        L    6
    UJP        L    8            !code produced by new pcompiler
    UJP        L    6
    UJP        L    9
L   6
    RETP
L   4=             8
```

!code produced by original pcompiler

Fig. 2.1 Effect of Modified CASE statement.

The modification done to the case statement handling can be appreciated by studying Fig. 2.1.

The case bounds are now checked by the single instruction CHKS, and in case of a range error, the execution is aborted. For values of I equal 1 and 3, UJC, instruction is generated. So that, if the values 1 or 3 occur at run-time, the execution is aborted instead of passing it as a no-operation.

2.2   THE PASCAL-P COMPILER OPTIONS

The Pascal-P compiler uses three external files. They are named INPUT, OUTPUT and PRR and are all of type TEXT. The file INPUT contains the source program; the file OUTPUT contains the source program listing (optionally), the symbol tables (optionally), and compiler messages; and the file PRR optionally contains the object program for the P-machine, in its assembly-language form. By 'optionally' we mean that these components may or may not be included in the respective files depending on the user's discretion. These user options can be set at any point in the user program by means of a pseudo-comment. A pseudo-comment is one in which the first symbol within the comment is a '$' symbol. Following the '$' symbol are option settings separated by ',' symbol. An option setting starts with the letters 'L', to indicate 'listing', or 'T', to indicate 'tables', or 'C', to indicate 'code', or 'R' to indicate generation of 'run-time tests' by

the compiler; this letter is followed by either a '+' or a '-' symbol, to turn the option ON or OFF respectively. Default settings for these options are : L is ON; T is OFF; C is ON; R is ON.

2.3   THE PASCAL-P COMPILER PERFORMANCE

The addition of the new instructions has enabled the compiler to produce a high density code. This not only results in saving of memory space, but also reduces the frequency of instruction fetches.

The performance of the compiler with a swappable code space has been studied, in a simulated environment on DEC-10. A memory manager was designed to map 5 physical segments into 14 virtual segments, for the purpose of the study. The replacement policy used is an LRU approximation. [see Chapter 3]. The compiler's speed has been observed to suffer due to a large number of segment faults.

An analysis of the compiler's procedure calling sequence [see Appendix 2] reveals that the number of inter-segment procedure calls are high. A reorganization of the P-code of the compiler was undertaken to cluster the related procedures and reduce inter-segment calls. The reorganization is based on an analysis of the procedure calling sequences of the compiler, for various test programs. A sample of the study results is available in Appendix 2.

The compiler's performance showed an appreciable improvement owing to the reorganization. The memory manager implemented in the PAS316 system, supports resident segments. By using this facility and making the most often used segments resident, the system performance can be improved further. This requires an indepth study of the compiler's procedure calling sequences.

CHAPTER 3

ENVIRONMENT AND SUPPORT FOR PASCAL-P SYSTEM

Nori et al. [13] describe the minimal configuration required to run a P-compiler through an interpreter. These requirements include more than 100K bytes of core memory and availability of a family of high level instructions for transfer of data between memory and external files.

A cursory look at the specifications of TDC-316 is enough to conclude that the machine does neither provide the requisite core nor the necessary file IO operators. Section 3.1 describes the methodology used for creating the high level operators for file handling as required by the PAS316 system. Section 3.2 investigates the various strategies that can be used to circumvent the lack of core memory and the scheme actually implemented.

3.1 ENVIRONMENT FOR PAS316 SYSTEM

The Pascal-P compiler supports the use of four text files. It needs a filing system to handle these files. Depending on the work environment, i.e., interactive or batch, the filing system must be augmented with the necessary IO support.

A Pascal program interacts with the environment only through the four predefined text-files. The PAS316 system
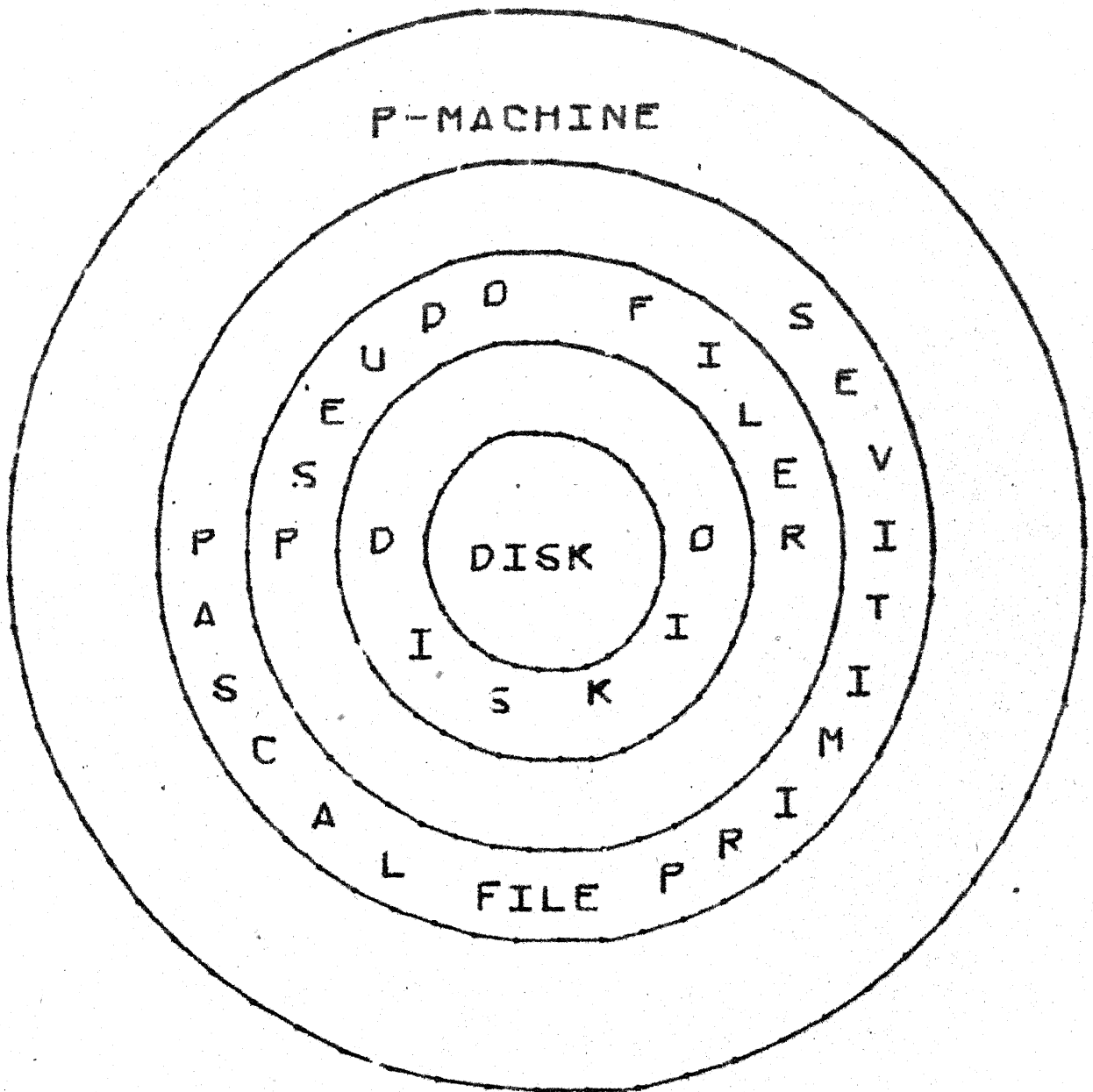
Fig. 3.1 P-Machine's interface with disk

expects all these files to reside on the disk. Since there is no filing system on TDC-316 at present, a temporary interface has been implemented. This interface is designed to support upto six files. In this system the user supplies the starting track, surface, sector address of his files. This parameter termed physical filename, for the purpose of this report may be replaced by the actual filename, when a regular filing system is available.

The PAS316 systems' interface structure with the disk is as shown in Fig. 3.1. A description of this interface follows :

Pascal-P machine as envisaged by Nori et al. accesses the files by means of a set of file IO operators. These operators are provided by the layer PAS316 file IO primitives. This layer enables the P-Machine to manipulate sequential files as if they are the fundamental structure of data handling on the mass-storage device. The transformation between the above view and the reality in which the data is stored as collection of smaller units of information called blocks is provided by the layer pseudo filer. The pseudo filer layer relies on diskio layer for transfer of data between computer memory and disk.

3.1.1 PAS316 - File Primitives [14]

The entire PAS316 system manipulates sequential files using the primitives provided by the module PAS316 file IO.

This uniformity between the P-code instructions operation and the rest of the system could be achieved mainly because of the nonavailability of explicit IO instructions in BLISS11!

A formal definition of a sequence is necessary to understand the working of the elementary file operators and is given below:

(a) $\langle \rangle$ denotes the empty sequence

(b) $\langle x_o \rangle$ is called a singleton   sequence

(c)   If $x = \langle x_1, \ldots, x_m \rangle$ and

$y = \langle y_1, \ldots, y_n \rangle$ are sequences, then

$x \,\&\, y = \langle x_1, \ldots, x_m, y_1, \ldots, y_n \rangle$

is the concatenation of x and y.

(d)   If $x = \langle x_1, \ldots, x_n \rangle$ is a non-empty sequence, then

$first\,(x) = x_1$

denotes the first element of x.

(e)   If $x = \langle x_1, \ldots, x_n \rangle$ is a non-empty sequence, then

$rest(x) = \langle x_2, \ldots, x_n \rangle$

Pascal allows only sequential access to files. The essence of sequential access is that at any time only a single specific component of the sequence is immediately accessible. This component is specified by the current position of the access mechanism. This position can be

altered using file operators. The most frequently used operation is to move to the next component of the file. Another operation of interest is to move the current pointer to the beginning of the entire sequence.

Also, a file can only be in one of two states : either in the state of being constructed (written) or of being scanned (read). We can formally define the position of current pointer of a file by regarding a file x to be partitioned into two parts, part x_L to the left of this pointer and a part x_R to its right. Hence,

$$x = x_L \not\!\!k \quad x_R \text{ (concatenation)}$$

Associated with each Pascal file is an implicit auxiliary variable that represents a buffer. This variable has the same type as the base type of the file. Since Pascal-P supports only text files, all buffer variables are of type character. The buffer variable of a file x is denoted by $x\uparrow$.

Pascal provides the following elementary file operators to its users.

i)   Constructing the empty tfile.

rewrite $(x) = x:= \langle \rangle$

The operator rewrite is used to erase any existing file x and initiate the process of constructing a new file, with name x.

ii) Extending a file

$$Put(x) = x := x \,\&\, x\uparrow$$

The operator put appends the value of x to the file x.

iii) Initiation of a scan

$$reset(x) = x_L := <\ >\ ;$$

$$x_R := x\ ;$$

$$x := first(x)$$

The operator reset is used to initiate the process of reading a file. It sets the current pointer for the file to its first component.

iv) Proceeding to the next component

$$get(x) = get(x_L\,\&\,x_R) = x_L := x_L\,\&\,first(x_R)\ ;$$

$$x_R := rest(x_R);$$

$$x := first(rest(x_R)).$$

note : first(s) is defined only if $s \neq <\ >$.

v) The predicate

$$eof(x) = x_R = <\ >$$

The operation get(x) can therefore only be executed if the predicate eof(x) is false.

To ease the task of handling text files, the Pascal user is also provided with the following additional operators.

vi)   read(x,VAR,v) = v:= x↑ ; get(x)

prerequisite: not eof(x)

vii) Write (x,e); e is an expression of type character

= x↑ :=e; put(x)

prerequisite: eof(x)

The line marker used is the ASCII character sequence CR LF.

viii) Writeln(x) append a line marker to the file x.

ix)   Readln(x) skip characters on file x upto the one immediately following the next line marker.

x)   Eoln(x).

A boolean function. True, if the file position had been advanced to a line marker; false otherwise. Also, if eoln(x) is true then x↑ = blank(ASCII 40B).

We shall now investigate the functions provided by each of the layers shown in Fig. 3.1.

3.1.2 PAS316 - File IO

This module enables the programmer to perform the elementary file operations as defined in the previous section. The user interacts with his files through a logical filename. For the purpose of this implementation the logical filename is required to be an integer between 0 and 5. The physical filename now consists of three integers denoting

the track, surface and sector address of the file on disk. This parameter can be replaced by a physical filename when a regular filing system is operational.

The user has to supply a one character buffer, for each of his files. This constitutes the auxiliary variable $x\uparrow$ defined in Section 3.1.1.

The main data structure manipulated by the file operators is

```
filetable : ARRAY[0..5]  OF (* six files *)
                RECORD
                    fsp: BYTE; (* file specifier returned
                                    by filer *)
                    eoln: BOOLEAN;
                    eof : BOOLEAN;
                    auxch  : BYTE; (* needed for managing
                                        EOLN lookahead *)
                    bufadr : WORD (* user supplied one
                                        character buffer *)
                END;
```

This module interacts with the pseudofiler described in Section 3.1.3.

The entry points of this module are :

Legend : l = logical filename;

b = Buffer address;

p = physical filename.

a)   reset (l,b,p);

b)   rewrite (l,b,p);

c)   get (l);

d)   put(l);

e)   readln (l);

f)   writeln (l);

g)   read (l,VAR ch);

h)   write (l,ch);

i)   readint (l,VAR N);

j)   endofline (l);

k)   endoffile (l);

l)   fileclose (l);

For details refer the program listing provided in Appendix 6.

3.1.3 Pseudo-filer

The pseudo-filer is nothing but a buffer manager, handling six buffers. It has an array of six buffers. A buffer is allocated to a file when it is opened and deallocated when a file is closed.

The data structures handled are :

```
buffertable : ARRAY[0..5, 0..255] OF BYTE;
openfilestable: ARRAY[0..5] OF
                RECORD
                baseadr: WORD; (* base disk address *)
                curadr : WORD; (* current disk address *)
                purpose: BYTE; (* read/write)*)
                bufferptr: BYTE (* current byte pointer *)
                END;
```

A file can be opened for either reading or writing, by giving the physical file name. The module returns a file specifier (fsp) for each file opened. This file identifier can be used for accessing the file by the user.

The entry points of the pseudo-filer are :

a)   open (physical filename, purpose, VAR fsp, VAR flag);
If flag = true is returned implies buffers exhausted.

b)   Close (fsp);

c)   getbyte (fsp, VAR ch);

d)   putbyte (fsp, ch);

3.1.4 Disk - IO

The pseudo-filer uses the disk-IO to put and get sectors from the disk.

This module is totally device dependent and interacts with the Disk controller of TDC-316 to transfer a given number

of words between the disk and memory.

The routines that perform the transfer are :

a)   putblk (track, surface, sector, word count, memory address)

b)   getblk (track,surface,sector,word count,memory address);

3.2   MEMORY MANAGEMENT

Clearly the memory requirements of a Pascal-P system
are too large to implement on the basic TDC-316 computer.
Some form of overlay technique is essential. There are a
number of ways of tackling this problem.

3.2.1 Multi-pass Compiler

Writing a multi-pass Pascal Compiler ! This as is
obvious will call for major changes in the design philosophy
of the existing Pascal-P compiler. Further, it will cost us
the elegance and simplicity of the P-compiler.

3.2.2 Code Segmentation

An alternative solution is to use virtual memory
techniques using Pascal procedures as virtual memory segments.
This approach is attractive as it avoids the error prone
alteration of the compiler code!

In this scheme, the P-code can be split into segments
by the compiler, each segment consisting of the code of one
procedure. Segmentation of the data segments would call for
considerable effort.

Hence, a proposal to restrict the virtual memory operations to the pure code block of the P-system and to use in core memory for the other segments has the following advantages:

a) Since the swappable segments contain pure code they need not be copied on the backing store on swapping.

b) Since a goto statement leading out of a procedure body is not available in the Pascal-P language, access to a segment occurs only upon procedure entry and exit. Hence, segment management considerations have to be taken care only on the CALL and RETURN P-instructions.

c) No interference to data accessing and addressing is necessary, which normally is cause for excessive over-heads, if performed through software.

This proposition is attractive for machines, lacking in hardware facilities for memory management, and suffering from lack of large address space. (e.g. 8080AµP). We, therefore, analyse this scheme in greater detail.

The memory management is likely to be complex as proce-dures vary in size. This also leads to external fragmentation. Further, as the replacement policy has to be implemented exclusively through software, it will involve considerable overheads. Thus, leading to considerable slackening of an already slow interpreter.

For a machine with a main memory of 64K bytes, after satisfying the memory needs of the various resident segments, the swap space available will be relatively small. We predict severe thrashing in such environments. This problem is further compounded by the structure of the compiler.

Owing to the recursive descent nature of the compiler, the number of active procedures is generally large. In addition, the most often used procedures like lexical analyzer and expression parsing routines are comparatively large.

Example 3.1 :   (Failure of strict LRU policy)

If we use an LRU replacement policy, because of the lack of swap space and due to the considerable variation in the size of the segments [see Appendix 2], we often have the problem of deleting a number of segments to bring in one segment.

Let the following table indicate the segments present in memory in order of decreasing age.

Segment	Size in units/blocks of memory
S0	4
S1	6
S2	1
S3	15

Table   3.1

```
PROCEDURE expression;
  PROCEDURE simpleexpression;
    PROCEDURE term;
      PROCEDURE factor;
        BEGIN
          .
          .
        END; (* factor *)
      BEGIN (* term *)
        factor;
        WHILE sy = mulop DO
          BEGIN
            .
            .
            factor;
            .
            .
          END
      END; (* term *)
    BEGIN (* simpleexpression *)
      .
      .
      term;
      .
      .
      WHILE sym = addop DO
        BEGIN
          .
          .
          term;
          .
          .
        END
    END; (* simpleexpression *)
  BEGIN (* expression *)
    simpleexpression;
    IF sy = relop THEN
      BEGIN
        .
        .
        simpleexpression;
        .
        .
      END
  END; (* expression *)

Fig.  3.2  Structure of Procedure Expression
```

If now a request to swap-in a segment of size 14 units is issued. All the 4 segments get removed.

To circumvent the problem dealt in Example 3.1, one may be tempted to device a strategy that deletes the largest segment of the eligible segments. This policy is also not sound enough.

Example 3.2 (Thrashing during expression parsing)

Assume the expression being parsed is

A + B

The call traffic to parse the factor A is as shown in Table 3.2. The structure of the procedure expression is summarised in Fig. 3.2. Each procedure in the expression block is relatively large.

Swapped out	Swapped in Expression	Call
Expression	Simple expression	Call
Simple expression	Term	Call
Term	Factor	Call
Factor	Term	Return
Term	Simple expression	Return
Simple expression	Expression	Return

Table 3.2

```
PROCEDURE insymbol;
  BEGIN (* insymbol *)
    CASE ch OF
      (*letters*).:BEGIN
              (* code for identifiers and reserved words *)
                END;
      (*digits*)..:BEGIN
              (* code for integer and real constants *)
                END;
      ''''        :BEGIN
              (* code for string constants *)
                END;
      (*etc*)
    END (* case *)
  END; (* insymbol *)
```

Fig. 3.3 Structure of Procedure Insymbol showing mutually
          exclusive code

```
PROCEDURE insymbol;
  PROCEDURE inident;
    BEGIN
    (* code for scanning identifiers and reserved words *)
    END;
  PROCEDURE innumber;
    BEGIN
    (* code for scanning number denotations *)
    END;
       .
       .
  BEGIN (* insymbol *)
    CASE ch OF
      (* letters *)...:inident;
      (* numbers *)...:innumber;
       .
       .
    END; (* case *)
  END; (* insymbol *)
```

Fig 3.4 : Restructured procedure Insymbol

```
PROCEDURE expréssion(...);
   PROCEDURE simpleexpression(...);
      PROCEDURE term(...);
        PROCEDURE factor(...);
          PROCEDURE facident(...);
            BEGIN
              (* code to handle identifiers *)
            END;
          PROCEDURE facset;
            BEGIN
              (* code to handle sets *)
            END;
          BEGIN (* factor *)

             CASE sy OF
               ident : facident;
               stringconst : facstring;
               lbrack : facset;

             END; (* case *)

          END; (* factor *)
        PROCEDURE mulfac;
          BEGIN
            (* handle <mulop><factor> *)
          END;
        BEGIN (* term *)
          factor(...);
          WHILE sy = mulop DO mulfac;
        END; (* term *)
      PROCEDURE addterm;
        BEGIN
          (* handle <addop><term> *)
        END;
      BEGIN (* simpleexpression *)

        term(...);

        WHILE sy = addop DO addterm;
      END; (* simpleexpression *)
      PROCEDURE relsexpr;
        BEGIN
          (* handle <relop><simpleexpression> *)
        END;
   BEGIN (* expression *)
     simpleexpression(...);
     IF sy = relop THEN relsexpr;
   END; (* expression *)
```

Fig. 3.5   Restructured Procedure Expression

It was observed that large parts of code are loaded and after execution of only a few instructions get swapped out.

Thus, some efficiency can be obtained by breaking the larger procedures (especially most often used, see Appendix 2) into small procedures. Two obvious candidates for this operation are Insymbol and Expression block. A simple and effective strategy for partitioning programs was suggested by Hurst [10].

This strategy is based on the observation that the procedures usually have mutually exclusive sets of code. Fig. 3.3 illustrates sets of mutually exclusive sections of the procedure insymbol. Hurst suggests that these mutually exclusive sections be used to define new procedures. Fig. 3.4 and Fig. 3.5 respectively illustrate the application of this strategy on procedures insymbol and expression.

Example 3.3 :

The effect of applying the above sub-division was studied on the procedure Insymbol. During the compilation of the binary tree traversal program shown in Appendix 2, it has been observed that procedure Insymbol was called 398 times and procedure Innumber was not called at all!

3.3 PAS316 MEMORY MANAGEMENT

We for the purpose of this project have not pursued the

strategy of Section 3.2.2, because of its excessive cost.
We instead, decided to exploit the additional hardware
features of TDC-316 to build a rudimentary virtual memory
manager. To this end we describe the relevant hardware
on TDC-316, in the next Section.

3.3.1 TDC-316's Memory Allocation and Protection (MAP) [7]

The TDC-316 MAP unit provides hardware facilities for
memory management for systems where the system memory size
is greater than 28K words.

The basic characteristics of the MAP unit are :

i)   16 user mode memory segments
ii)   16 superviser mode memory segments
iii) 8 segments each for instructions and data in each mode
iv)   Segment lengths from 32 to 4096 words.

Eventhough, the wordlength and operational logic of
TDC-316 is of 16-bit length, the CPU and Bus addressing logic
are actually of 18-bits. These extra two bits provide the
basic framework for expanded memory operation.

The MAP unit makes available in the supervisor and
user mode, 56K words of virtual address space (28K of
Instruction and 28K of data). However, this would require
that the programs resident in the I-space be pure. If not,
D-space cannot be enabled and only 28K words of virtual
space will be available in each mode.

Even this extended memory facility cannot directly be used for the Pascal-P system. Pascal-P needs data space in excess of what is available. The P-code interpreter regards both the P-code and the data segment as data. Thus eventhough, an addressing capability of 56K words is adequate, 28K words of data space is inadequate.

We decided to use the MAP unit and provide the required data space by using the Transfer Communication Mode instructions of TDC-316. This enabled us to use the hardware support available for memory management.

This decision costs us execution of 2 additional instructions for each fetch cycle of the P-machine (i.e. for each access to the P-machine's code array). The scheme implemented is shown in Fig. 3.6.

Supervisor memory   User memory

I:instruction

D:data

U:user

S:supervisor

| Traps, memory manager, interpreter etc. | I-space | Free | I-space |

| Interpreter data+tables, Pascal data segment | D-space | Pascal code segment | D-space |
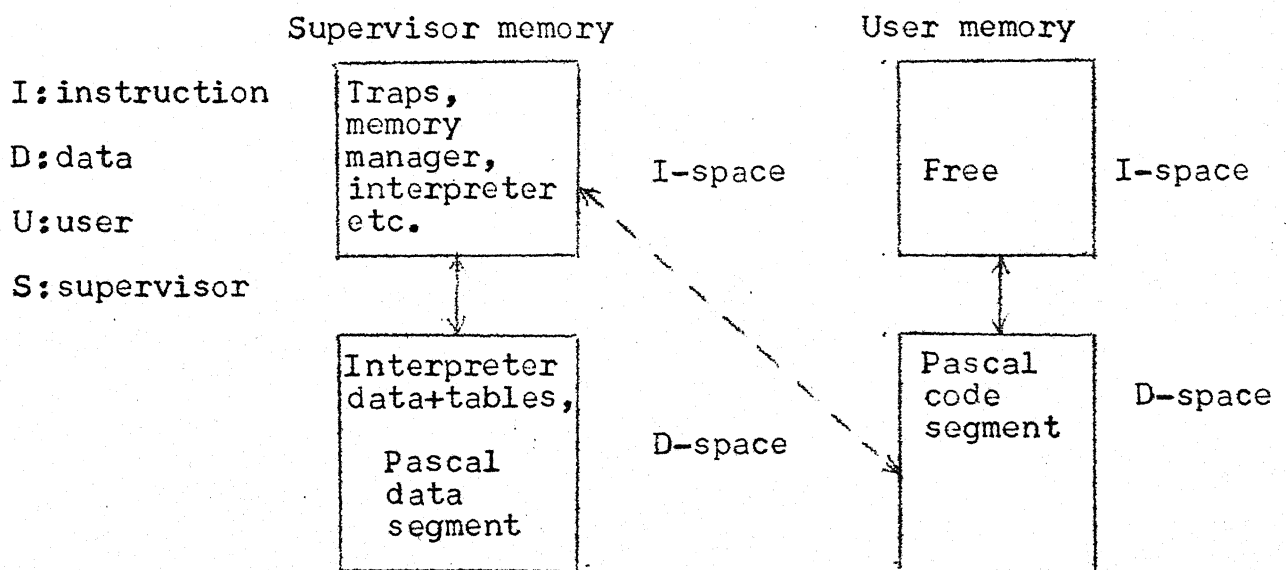
Fig. 3.6 PAS316 Memory Allocation Scheme

```
PROCEDURE    fetch (VAR   v);
   BEGIN
      transfer communication mode to FROM UD space;
      v:= pcode [ppc];
      transfer communication mode to NORMAL
   END; (* fetch *)
PROCEDURE deposit (v);
   BEGIN
      transfer communication mode to TO UD space;
      pcode [ppc]:= v;
      transfer communication mode to NORMAL
   END; (* deposit *)
```

3.3.2 Segmentation

The 32K words of total addressable area in each of the
4 spaces, is divided into 8 segments. The segments can
vary in length   from 32 words to 4K words in multiples of
32 words. Thus, there are a total of 32 segments.

Each of the 32 segments has a relocation constant
(MAP-AF), a length constant (MAP-LF) and an access control
specifier (ACF) associated with it. These are stored in a
register set called Active Segment Register (ASR), file.

3.3.3 MAP Operation

When the MAP unit is enabled, all addresses generated
by the CPU are routed through it. The current processor

status specifies one of the four spaces, while the
segment selection is through the three most significant bits
of this virtual address. The relocation constant is
combined with the 16-bit virtual address to get a relocated
18-bit physical address. At the same time the segment
length and the access type are checked against the length
constant and access control specifier for validity. If the
access is a valid one, the relocated address is put on the
bus for data access. Otherwise, the relevant information
is strobed into the MAP status registers and access is
aborted and a trap is generated. The status registers
contain relevant information for program recovery and roll
back.

3.3.4 Segment Manager

Since the Pascal-P code and Data Segments are contiguous
arrays the maximal allowed segment size of 4K words was chosen.

Thus the memory manager has to handle the 32 virtual
segments in 4 spaces, through swapping in the 8 pages of core
available on TDC-316. As the 8th segment in each space is
required to be set aside for register and ROM use, we
essentially have 28 virtual segments and 7 physical pages each
of 4K words.

The memory manager has a fixed 128K word swap space on
the disk to accomodate the 32 virtual segments. The swap
space has been organized such that the disk address calculation

is simple and once the transfer of data begins, there is
no additional seek delay. The swap space has been organized
as follows :

The swap space for each segment is on a separate track.
Hence, mapping from virtual segment number to track address
is one to one. In each track four surfaces are used, each
surface stores 1K words. The eight sectors needed to store
1K words have been chosen such that, all the 1K words can be
transferred at a time through DMA, and when the surface is
switched the free sectors pass under the head. Thus, even
latency delays are minimized.

3.3.5 Replacement Policy : Passing the Parcel [12]

The segment replacement is based on an LRU approximation.
It can best be described as Not Used Recently (NUR) policy.

The policy can be best understood by visualizing the
segments present in core to be involved in the passing the
parcel game. At any instant of time a segment which is in
core will be holding the parcel. A segment holding the parcel,
is considered as the next candidate for swap out by the manager.
It is actually marked OUTCORE, even though is in-core.

When a trap occurs (music stops!), if the reference is to
the parcel holder, then it passes the parcel and is marked,
INCORE. This case is called a dummy trap, because no swapping
of segments has actually taken place. If the reference was

not to the parcel holder, it is swapped out and the necessary segment is brought in.

Physical seg.No.	Virtual seg.No.	Additional inf.
0	0	
1	1	
2	8	
3	31	
4	6	
5	22	
6	24	
7	7	

Parcel holder [1]

Fig. 3.7   Core table

```
ALGORITHM   swappolicy;
    VAR
        coretable : ARRAY[0..7]  OF (*8physical segments *)
                    RECORD
                    segno       0..32; (* 32 virtual segments *)
                    otherinfo
                    END;
        parcelholder : 0..7;
```

```
PROCEDURE  passparcel;
  BEGIN
      give parcel to the nearest eligible neighbour;
      (* round Robin *)
      (* a facility to make some segments permanently resident
      is available.  These segments never hold the parcel *)
      make coretable [parcelholder]·segno OUTCORE
  END; (* passparcel *)
PROCEDURE   initpolicy;
  BEGIN
    FOR I:= 0 TO 7 DO
       Coretable [I]·segno:= 32; (* invalid segno *)
    parcelholder:= -1; passparcel
  END; (* initpolicy *)
PROCEDURE  system-wants (seg);
  (* segment fault has occurred. System needs the segment
  SEG *)
  BEGIN
    IF coretable [parcelholder]·segno ≠   seg  THEN
        BEGIN
           Swapout (parcelholder);
             Swapin(seg,parcelholder)
        END;
    make coretable [parcelholder]·segno INCORE;
    passparcel
  END; (* system-wants *)
```

3.4 CONCLUSIONS

We have described the strategies used to provide the requisite environment for running a Pascal-P system.

In Appendix 2, we show the performance of the memory manager. The relation between the trap rate and the procedure calling sequence has been studied.

The Pascal file operators implemented have been used uniformly throughout the implementation. These were used as the IO primitives for the rest of the system.

CHAPTER 4

P-ASSEMBLER AND P-INTERPRETER

The P-compiler transforms a Pascal program into an equivalent P-Machine's assembly language program. The P-Assembler assembles this assembly language program. The syntax of the P-Machine's assembly language is given in Appendix 4. The P-Assembler loads the P-Machine's code array PCODE with the object code, ready for interpretation. The object code can be salvaged into an object file by using the PSAVE program.

The P-interpreter interprets the object code available in the code array PCODE. It can accept input from two standard files INPUT and PRD and can generate results in the two standard files OUTPUT and PRR.

4.1 P-ASSEMBLER

The P-assembler is a single pass assembler with back patching. It accepts the source program in file PRR and loads the object code in the P-machine's code array PCODE. It uses a mnemonic table and a labeltable. The label table is used for storing the addresses of tables used in the assembly language and for back patching.

4.1.1 Data Structures

```
TYPE
    labelrange = 0 .. max-no-of-labels
    instrange  = 0 .. max-no-instr;
    spfcnrange = 0 .. max-no-spfcns;
    labst = (entered,defined); (* label status *)
    alfa  = PACKED ARRAY [1..4] OF CHAR;
VAR
    inst-table : ARRAY[instrange] OF alfa;
    (* binary search performed to locate mnemonic *)
    special-fcnproc-table : ARRAY[spfcnrange] OF alfa;
    (* linear search to locate function/procedure mnemonic *)
    opcode-table: ARRAY[instrange] OF INTEGER.
    (* opcode stored,correspond to mnemonics in instruction
    table *)
    (* label table *)
    labelvalue: ARRAY[labelrange] OF INTEGER;
    label-status: ARRAY[labelrange] OF labst;
    Pcode: ARRAY[0 .. maxcode] OF BYTE;
```

4.1.2 Algorithm

The generation of the program from the BNF definition
of the Assembly language defined in Appendix 4 is relatively
straight forward. Since the number of instructions of
opcode class 1 are relatively large, the P-assembler resolves
the opcode class using a CASE statement rather than storing
this information in the opcode table.

```
ALGORITHM    P-assembler;
  BEGIN
    init;
    generate;
    ppc:= O;
    generate
  END; (* P-assembler *)
```

Generate produces the object code for an assembly record.

```
PROCEDURE generate;
  BEGIN
    WHILE NOT eoln (source-file) DO
      BEGIN
        process-assembly-statement;
        Readln (source-file)
      END
  END; (* generate *)
PROCEDURE process-assembly-statement;
  BEGIN
    CASE first character of line OF
      'I' : ignore the line;
      'L' : update-label-table;
      ' ' : assemble-the- instruction
    END
  END; (* process-assembly-statement *)
```

```
PROCEDURE update-label-table;
   BEGIN
      i) get the label value, which is either the current value
         of location counter (ppc) or is as defined in this
         source line;
      ii)If there was a forward reference of this label then
         traverse the code array and update;
      iii)Store value of label in the label table
   END; (* update-label-table *)
PROCEDURE assemble-the-instruction;
   BEGIN
      getmnemonic;
      binary search instruction table and get
         corresponding opcode value from opcode table;
      CASE opcode class of instruction OF
         procedure call instruction              : ... ;
         return instruction                      : ... ;
         load constants                          : ... ;
         special instr. CHK                      : ... ;
         special instr. MST                      : ... ;
         opcode class 2                          : ... ;
         opcode class 3                          : ... ;
         opcode class 4, ENT instruction         : ... ;
         opcode class 5                          : ... ;
         opcode class 6                          : ... ;
         opcode class 7                          : ... ;
         opcode class 8                          : ... ;
         others (opcode class 1)                 : ... ;
   END
```

The transformation from the assembly language to the P- machine's instructions given in Appendix 5 is straight-forward except in the following cases :

Assembly Instruction	Code generated
LDCB O	LDCBF
LDCB 1	LBCBT
LDCN nilvalue	LDCN
CSP special fcn name	Specialfcn name
Eg: CSP WRI	WRI

4.2 P-INTERPRETER

4.2.1 P-Machine's Architecture

The P-machine consists of 7 registers and a memory. The registers are

PPC - P-machine's program counter

PSP - P-machine's stack pointer

MP - Mark pointer, points to the base of the current activation record

NP - New pointer, points to the top of the Heap

OP - opcode

P - P-operand

Q - Q-operand.

OP,P and Q together from the Instruction Register.

The memory is organized as two linear arrays. One called PCODE and the other PSTK. The PCODE is an array of bytes which is accessed through the two procedures fetch and deposit. The PSTK is an array of words. PPC is an index into PCODE and PSP, MP and NP are indices into PSTK.

The Stack and the Heap as required by the executing Pascal program grow towards each other starting at the two ends of PSTK.

The stack grows from O upwards and consists of all directly addressable data according to the data declaration. Storage overflow occurs if PSP and NP meet. The Heap grows downwards from the top of the PSTK.
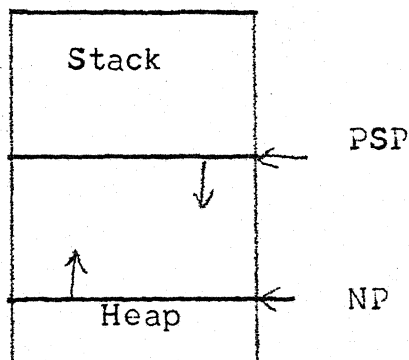
Fig. 4.1 PSTK

The Heap grows when the memory allocation is done through the standard procedure NEW.

The Stack consists of a sequence of activation
records, each belonging to an active procedure or function.
The activation record starting at PSTK [0] belongs to the
main block of the Pascal program.

An activation record consists of the following sequence
of information.

```
┌──────────────────────────┬──────────────
│  Function return         │        ↑
│      Value               │
├──────────────────────────┤
│      Static Link         │
├──────────────────────────┤         Mark stack
│    Dynamic Link          │
├──────────────────────────┤
│   Return Address         │        ↓
├──────────────────────────┴──────────────
│   Parameter Section      │
├──────────────────────────┤
│   Local Variables        │
├──────────────────────────┤
│    Temporaries           │
├──────────────────────────┤
│   Expression Stack       │
└──────────────────────────┘
```
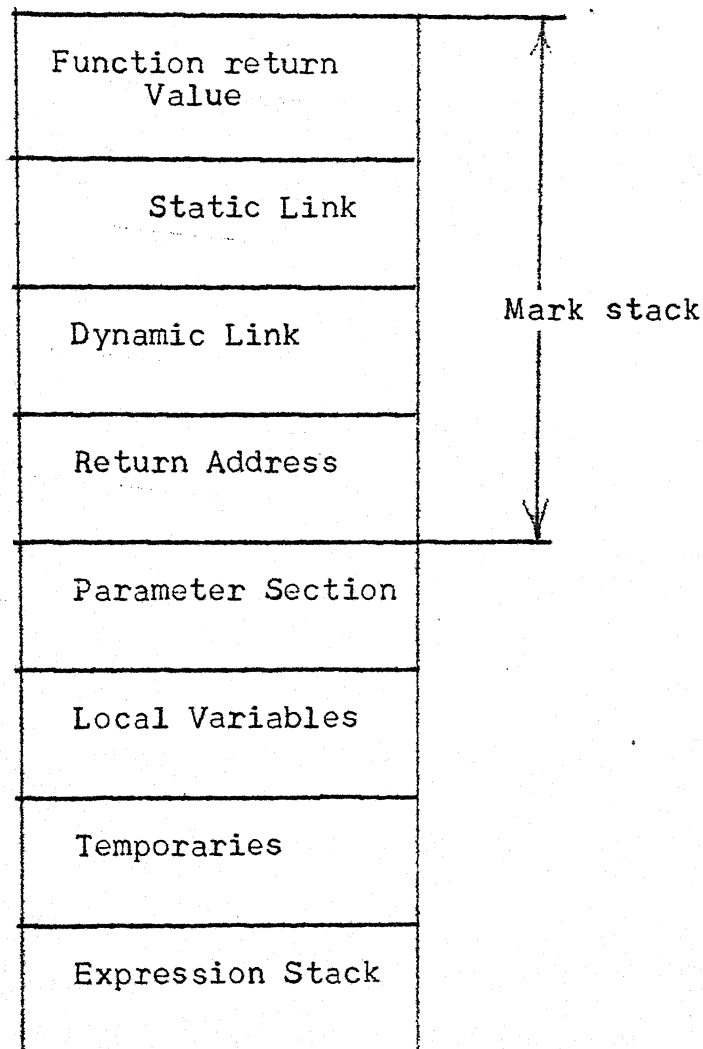
Fig. 4.2   Activation Record

a <u>mark stack</u> part; a <u>parameter</u> section if there are any parameters to the procedure or function to which the activation record belongs; a <u>local-variables</u> sections if there are any local variables declared within the procedure or function to which the activation record belongs; any <u>temporary</u> elements which may be required in the program-evaluation process.

The mark stack paft has further internal structure. It is 5 words long. The first field is space for preserving the value returned by a function. It is included even for procedures for the sake of uniformity. This field is of 2 words, to enable returning real values. The second field is a pointer called static link; the third field is a pointer called dynamic link; the fourth field is a pointer called the Return address. The static and dynamic links are indices into PSTK and return address is an index into PCODE.

There are two types of parameters :

a) pointers (into PSTK) in case the corresponding parameters are of type call by reference or of type call by value but of record or array type or

b) the parameter is call by value and the value is passed.

Before a procedure or function call is made, a mark stack instruction (MST) is executed with a parameter which allows the links to be filled.

Then a series of expression evaluations fill in the parameter section. After this a call user procedure (CUP) or a call standard procedure (CSP) instruction is executed.

4.2.2 P-Machine's Instruction Set and Addressing Modes

The P-Machine's Instruction set is given in detail in Appendix 5. The constructs defined in Pascal are strongly reflected in the instructions of the P-machine.

A set of 4 new instructions have been introduced into the P-machine, to supplement those defined in the original P-compiler [13]. These are LDL, STL,,LAL and UJC.

The introduction of new instructions to load and store local variables is based on the premise that variables local to a procedure are accessed more frequently and therefore require fast access by short instructions, whereas access to remote variables is relatively rare and requires less efficiency.

The LDL, STL and LAL are new mnemonics for the original P instructions LOD 0, Q; STR 0, Q; and LAL 0, Q with the parameter 0 dropped.

UJC, is used to trap undefined case labels at run time. The compiler generates the UJC instruction for all values of the case variable, that have not been defined but lie within the range of the case statement (Refer Fig. 2.1.)

Hence, access to the most often used variables, which in general are the local variables and global variables (variables defined in the outermost block of the Pascal program) are accessed through base registers. The base of the current activation record is stored in the register MP and the base of the outermost block is 0.

Instructions consist of one or several bytes. They can be divided into four basic categories : load and store instructions, operators, control instructions and miscellaneous instructions :

The load and store instructions transfer data between memory (stack or heap) and the top of the stack, where they are accessed by operators. The top of the stack, where data are loaded as intermediate results is also called the expression stack.

The system allows the creation of an activation record or a dynamic variable, only if there are atleast 100 words available, for the expression stack. This feature is based on the premise that the amount of stack space needed for evaluation of an expression is less than 100 words, which is generally true. Load and store instructions involving local and global variables require only a single offset address because the stack address is implicit. For intermediate variables, an additional parameter specifies the level difference, which indicates, the number of levels to descend

using the static links for accessing the variables. The
load and store instructions are further subdivided depending
on the data size. The transferred data may be a word
(16 bits), a double word or four words.

The addressing modes available on the P-machine are
defined as follows :

(p and q denote the instruction parameters, 'a' denotes
the resulting address and psp is stack pointer) :

-    local mode : a = MP+q, used for variables local to
     procedures.

-    Global mode : a = q, used for global variables.

-    Stack mode: a = base (p) + q, used for intermediate
     variables.
     Base (p) returns the contents of the dynamic link
     field of the activation record which is p levels below
     the current activation record. Base descends the stack
     using the static link fields of activation records.

-    Indirect mode : a = pstk [psp] + q, mode used for
     indirect addressing and access via pointers.

-    Indexed modes : a = s1 + q * s2, s1 is the arrays base
     address, and s2 the computed index (s1, s2 are on stack
     at pstk [psp-1] and pstk[psp]), q is a multiplier
     depending on the size of the accessed data type.

- immediate mode: the immediate mode is used to load constants. The size of the operands depends on the type of the constant being loaded. The operand size may be 0,1,2, 4 or many words depending on the constant being loaded is a boolean constant or nil, an integer, a real, a set or a string respectively.

The second category of instructions are the operators. They take operands from the top of the stack and replace them by the result. The P-machine instruction set includes operators for integer, real, boolean and set arithmetic. There is a full set of comparison instructions which generate a boolean result. Distinct sets are available for integer, real, boolean and set comparisons.

Control instructions include procedure calls, procedure returns and jumps. Conditional jumps are generated for IF, WHILE, REPEAT and FOR statements. Indexed jumps are generated for CASE statements. They fetch their operand from the stack. There are explicit instructions for checking the array index bounds and subrange bounds, and undefined case labels.

The miscellaneous instructions category contains operators for reading and writing data on files, marking stack, entering a block and dynamic memory allocation and deallocation.

4.2.3 P-Instruction Execution

For the purposes of execution, the opcodes are divided into three categories. The type of the opcode is detected by the interpreter by checking one bit of the opcode. The P-interpreter program is available in Appendix 6. The algorithms used for interpreting the various instructions are self-explanatory.

```
ALGORITHM  P-interpreter;
  BEGIN
    interpret: = true;
    WHILE interpret DO
      BEGIN
        fetchopcode;
        CASE type-of-opcode OF
        type 0 : execute;
        type 1 : fetchoperands; execute;
        type 2 : fetchoperands; set registers execute
      END
    END (* interpret is made false when a STP instruction
        is executed *)
END ; (* P-interpreter *)
```

CHAPTER   5

CONCLUSION

The P-Assembler, P-interpreter and the support
software have been written and implemented on TDC-316.
Except for a small module, all modules were written in
BLISS11. The remaining module concerned with restoring
of computer registers after a memory management trap, was
coded in assembly language, to have explicit control on
the code generated.

The developed software has not yet gone through exhaus-
tive performance evaluation because of the constraints of
time. But the limited experience we have is satisfying.

A user interacts with the system through the command
processor. He can use this processor to compile, assemble and
interpret his Pascal program. If he so desires he can direct
the command processor to skip any one or more of the above
steps. An user manual is available in Appendix 8. Fig. 5.1
indicates the structure of the PAS316 system.

5.1 CRITICAL REVIEW OF PAS316 AND SUGGESTIONS FOR FURTHER WORK

The major bottleneck in the use of the present system is
the nonavailability of an efficient filing system. An editor
to input Pascal programs and data is needed to make the system
fully operational. A filing system and a Text Editor are
currently under development in sister projects [1,3].
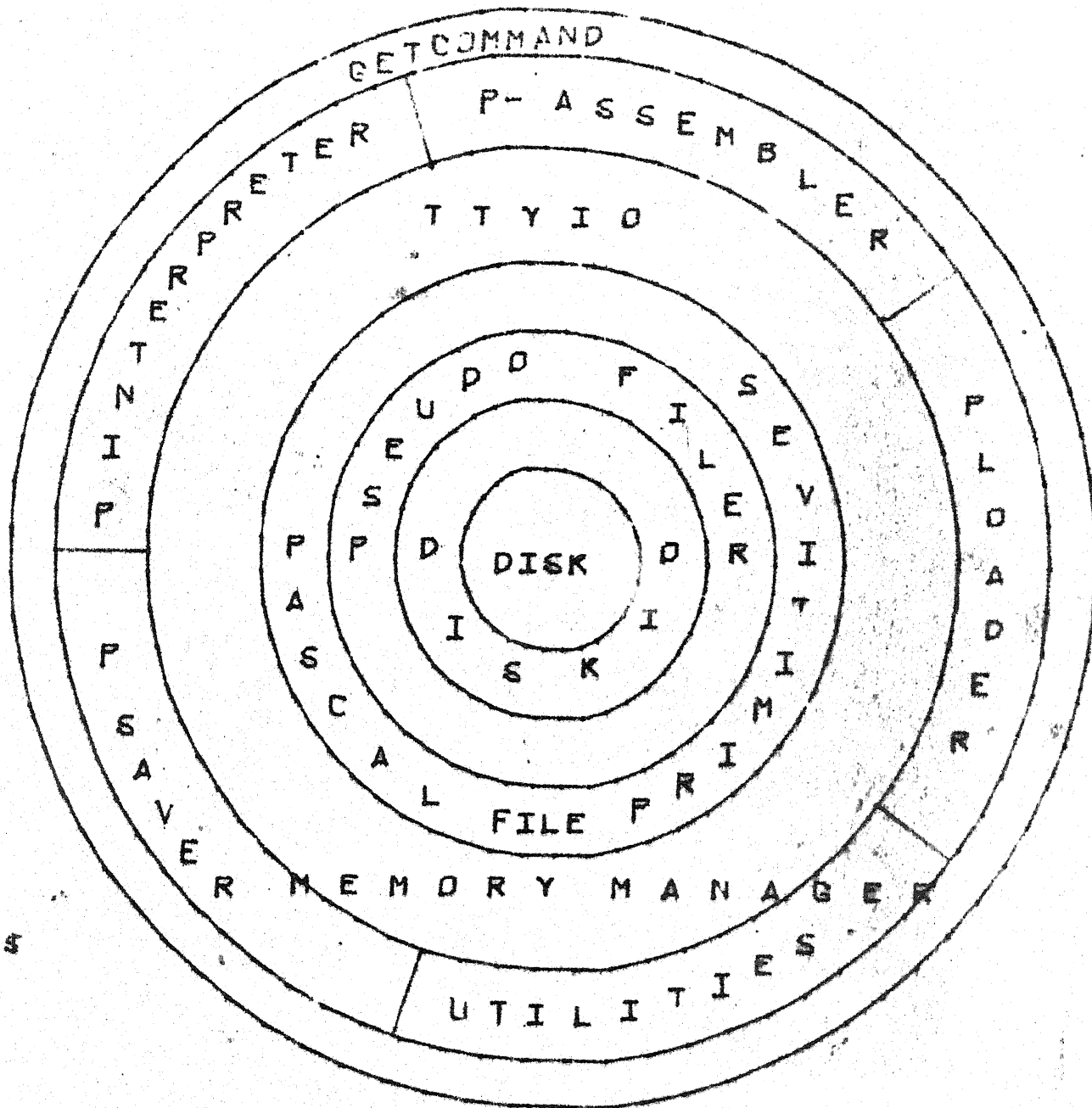
Fig. 5.1 Structure of PAS316 system

The present version of PAS316 does not support real arithmetic. A floating point processor or a floating point software package may be added to augment the system.

An integrated test of the complete P-system has not yet been made on TDC-316 because of lack of requisite utility systems. All the modules have been tested and found satisfactory on TDC-316. Also assembly and interpretation of Pascal programs have been successfully tested on a simulator of TDC-316 on DEC-10.

REFERENCES

1. Aarti Kumar, A filing system for TDC-316, Department of Computer Science, IIT-Kanpur, 1982, to be published.

2. Aho, A.V., and Ullman,J.D., Principles of Compiler Design, Addison-Wesley Publishing Company, 1979.

3. Ajay Tyagi, A Text Editor for TDC-316, Department of Computer Science, IIT Kanpur, 1982, to be published.

4. Berry,R.E., Experience with the Pascal P-compiler, Software - Practice and Experience, 8, No.5, 617-627, 1979.

5. Colemann, S.S., Pode, P.C., and Waite, W.M., The mobile programming system: JANUS, Software - Practice and Experience, 4, No.1, 5-23, 1974.

6. Digital Equipment Corporation, BLISS-11, Programmer's Manual, 1974.

7. Digital Equipment Corporation, PDP-11, Processor Handbook, 1971.

8. Eckhouse, R.H., Morris, R.L., Minicomputer Systems Organization, Programming and Applications (PDP-11), Prentice Hall Inc., Second Edition, 1979.

9. Electronics Corporation of India Limited, System Manual TDC-316, Volumes I,II and III.

10. Hurst, A.J., Pascal-P, Program structure and Program Behaviour, Software - Practice and Experience, 10, No.10, 1029-1036, 1980.

11. Jensen, K., and Wirth, N., Pascal – User Manual
     and Report, Springer-Verlag, Berlin, 1974.

12. Madnick, S.B, and Donovan, J.J., Operating Systems,
     McGraw-Hill Book Company, 1974.

13. Nori, K.V., Ammann, U., Jensen, K., Nageli, H.H.,
     Jacobi, Ch., Pascal-P Implementation Notes, Nr.10,
     ETH Zurich.

14. Wirth, N., Algorithms ʻ+ Data Structures = Programs,
     Prentice-Hall Inc., 1976.

15. Wirth, N., The Personal Computer, LILITH, Nr. 40,
     ETH Zurich.

```
          APPENDIX - 1
          ************
```

!The Language Pascal-P
****

    The language processed by the Pascal-P compiler is
'standard Pascal' with some omissions and one change.

The features which are not processed are:

(a) procedures/functions as parameters,
(b) goto statements leading out of procedure/function bodies,
(c) all kind of files except predefined character files:
    The pre-defined standard text files are:
       INPUT,PRD (input files)
       OUTPUT,PRR (output files)
(d) all features associated with 'packing'.

Change:

(e) The standard procedure 'dispose' is replaced by
    'mark' and 'release';
       MARK(P)   where p is of any pointer type, marks the
                      heap in the current state.
       RELEASE(P)   releases all items created by a NEW
                      instruction since the corresponding
                        MARK(P).

```
                    APPENDIX - 2
                    **************

!PASCAL-P COMPILER PERFORMANCE EVALUATION:
!*****************************************

!TEST PROGRAM USED FOR THE FOLLOWING TESTS:
!*****************************************

(*$1+,c+*)
(**binary tree traversal *)
PROGRAM traversal(input,output);
TYPE  ptr = ^node;
      node= RECORD
                   info :CHAR;
                   ll,rl : ptr
            END;

VAR  root :ptr; ch:CHAR;
     t:^INTEGER;

PROCEDURE preorder(p:ptr);
    BEGIN
        IF p<> NIL THEN
            BEGIN write(output,p^.info);
                  preorder(p^.ll);
                  preorder(p^.rl)
            END
    END;
    (*preorder *)

PROCEDURE inorder(p:ptr);
    BEGIN
        IF p<> NIL THEN
            BEGIN inorder(p^.ll);
                  write(output,p^.info);
                  inorder(p^.rl)
            END
    END;
    (*inorder *)

PROCEDURE postorder(p:ptr);
    BEGIN
        IF p<> NIL THEN
            BEGIN postorder(p^.ll);
                  postorder(p^.rl);
                  write(p^.info)
            END
    END;
```

RET	3607	93
CSP	7004	3105
TXA	3180	3152
EXI	6704	3927
NEO	1394	197
GEO	2728	
GKT		
LES	1977	
LES	1479	45
UJP	7755	117
FJP	2009	800
XJP	749	788
CKF	749	223
FJF	1857	9
ADT	3355	
ADR	1391	
SSF		
SGST		
FJD		
FJK		
NGTR		
NGGR		
SSDR		
SSQR		
ABT	4736	99
ABR	2047	537
MDT	2163	27
ANDR		
TOR	411	114
DLF	1597	181
TNT	183	
TNN		
MDD		
ODT		
MPT	335	
MPR	497	
DVT	2744	2125
DVR		2291
MOV	17652	
LCA	5555	1563
DEJ	759	1508
STP		148
LDD		37
SHD		
LAD		
(HJC)		
GET	0	0
PUF	0	
RSF	4	1
RLN	0	

NEW	58	81
WLN	57	291
WKS	68	533
FLN	1	1857
WRT	60	347
WKR	0	0
WRO	63	2026
RDR	0	0
RDC	0	0
SIN	1	1857
COS	0	0
EXP	0	0
LOG	0	0
SQT	0	0
AIN	0	0
SAV	1	4

The addition of new instructions and modification of existing one's has caused:

(a) Saving in CODE = 238 + 296 + 1557 + 505 + 148
= 2744 bytes.

(b) Saving in No. of Fetches =
2156 + 6952 + 17648 + 5553 + 759
= 33068

Procedure Call Statistics of the P-Compiler
****

Compilation of BINFRE.PAS

CALLING PROC.NAME	SIZE	CALLED PROC.NAME	SIZE	TIMES CALLED
SYSTEM	3	MAIN	106	1
MAIN	106	INITSCAL	54	1
MAIN	106	INITSETS	24	1
MAIN	106	INITTABL	12	1
INITTABL	12	RESWORDS	266	1
INITTABL	12	SYMBOLS	314	1
INITTABL	12	RATORS	116	1
INITTABL	12	INSTRMNE	312	1
MAIN	106	PROCMNEM	140	1
MAIN	106	ENTERSTD	104	1
MAIN	106	STDNAMES	212	1
MAIN	106	ENTSTDNA	495	1
ENTSTDNA	495	ENTERID	73	40
MAIN	106	ENTERUND	186	1
ENTERUND	186	GENLABEL	9	2
MAIN	106	INSYMBOL	401	1

INSYMBOL ... NEXTCH
NEXTCH ... ENDOFLIN
INSYMBOL ... OPTIONS
OPTIONS ... NEXTCH
INSYMBOL ... INIDENT
INIDENT ... NEXTCH
MAIN ... PROGRAMM
PROGRAMM ... INSYMBOL
PROGRAMM ... BLOCK
BLOCK ... INSYMBOL
BLOCK ... TYPEDECL
TYPEDECL ... INSYMBOL
TYPEDECL ... TYP
TYP ... INSYMBOL
TYP ... SEARCHID
TYP ... ENTERID
TYPEDECL ... FIELDLIS
FIELDLIS ... ENTERID
FIELDLIS ... INSYMBOL
FIELDLIS ... TYP
TYP ... SIMPLETY
SIMPLETY ... SEARCHID
SIMPLETY ... INSYMBOL
BLOCK ... VARDECLA
VARDECLA ... ENTERID
VARDECLA ... INSYMBOL
VARDECLA ... TYP
BLOCK ... PROCDECL
PROCDECL ... SEARCHSE
PROCDECL ... GENLABEL
PROCDECL ... ENTERID
PROCDECL ... INSYMBOL
PARAMETE ... PARAMETE
PARAMETE ... ENTERID
PROCDECL ... SEARCHID
BLOCK ... BLOCK
BODY ... BODY
BODY ... PUTLABEL
BODY ... GENLABEL
GENUJPEN ... GENUJPEN
BODY ... PUTIC
STATEMEN ... STATEMEN
STATEMEN ... INSYMBOL
IFSTATEM ... IFSTATEM
EXPRESSI ... EXPRESSI
SIMPLEEX ... SIMPLEEX
TERM ... TERM
FACTOR ... FACTOR
FACTOR ... SEARCHID
FACTOR ... INSYMBOL
EXPRESSI ... SELECTOR
LOAD ... LOAD
... GETADCH

LOAD	GFN1	141	34
GEN1 EXPRESSI	PUTIC	152	121
LOAD GEN2	INSYMBOL	261	4
GEN2 EXPRESSI	GFN2	141	132
COMPTYPE EXPRESSI	PUTIC	257	4
EXPRESSI	COMPTYPE	261	4
IFSTATEM	COMPTYPE	228	5
IFSTATEM	GFN2	261	4
GENFJP	GENLABEL	54	4
IFSTATEM	GENFJP	54	4
IFSTATEM	LOAD	500	5
STATEMEN	PUTIC	501	5
COMPOUND	INSYMBOL	54	4
STATEMEN	STATEMEN	194	135
STATEMEN	COMPORE	194	45
CALL	STATEMEN	194	43
	SEARCHID	140	54
	CALL	140	23
	INSYMBOL	307	23
	WRITE	401	13
	SEARCHID	307	12
	INSYMBOL	307	36
CALLNONS	EXPRESST		12
CALLNONS	LOAD	247	14
CALLNONS	SEARCHID	257	48
CALLNONS	LOAD	152	13
CALLNONS	GEN2	401	16
GENCUP	INSYMBOL	343	16
IFSTATEM	CALLNONS	343	32
BODY	GEN1	343	16
CALL	INSYMBOL	343	12
READ	EXPRESSI	343	28
READ	LOAD	41	16
VARIABLE	COMPTYPE	54	17
VARIABLE	GFNCUP	417	5
VARIABLE	PUTIC	417	35
READ	PUTLABEL	140	6
LOADADDR	INSYMBOL	168	1
READ	GEN1	168	1
READ	READ	168	2
CALL	SEARCHID	222	1
NEW	INSYMBOL	22	4
	VARIABLE	168	4
	SEARCHID	111	4
	INSYMBOL	168	1
	SELECTOR	168	7
	LOADADDR	140	2
	GEN1	152	3
	GEN1		1
	COMPTYPE		
	NEW		
	VARIABLE		

SELECTOR	450	GEN1	152	5
NEW	152	LOADADDR	111	1
NEW	152	GEN2	257	1
STATEMEN	152	GEN1	152	2
STATEMEN	194	ASSIGNME	150	2
ASSIGNME	150	SELECTOR	450	2
ASSIGNME	150	LOADADDR	111	2
ASSSIGNME	150	INSYMBOL	401	4
ASSIGNME	150	EXPRESSI	261	2
ASSIGNME	150	COMPTYPE	228	2
ASSIGNME	150	LOAD	141	2
STORE	75	STORE	75	3
STORE	75	GETADCH	62	
GENO	39	GENO	39	4
CALLNONS	343	PUTIC	27	1
IFSTATEM	544	LOADADDR	111	1
CALL	140	GENUJPEN	37	1
MARK	26	MARK	26	1
MARK	26	VARIABLE	22	1
MARK	26	LOADADDR	111	1
CALL	140	GEN1	152	1
RELEASE	26	RELEASE	26	1
RELEASE	26	VARIABLE	22	1
RELEASE	26	LOAD	141	1
WRITE	377	GEN	152	2
LOADADDR	111	LOADADDR	111	1
STRING	211	STRING	211	1
WRITE	377	COMPTYPE	228	1
BODY	417	GETCDF	41	1
BODY	417	GENO	39	1

!Performance of the P-Compiler with segmented CODE Space
****

Compilation of BI?RE.PAS

Segment Size : 1024?
No. of virtual segments = 15;
No. of physical segments = 5;
Segment Replacement Policy : Passing the Parcel.

Compiler in its Original form :

S i m u l a t i o n R E S U L T S
****

Total number of TRAPS: 1032

Number of Dummy TRAPS: 37?

Number of Swap TRAPS: 662

Segment No.	Access Count	SwapinCount	SwapoutCount
0	123657	85	84
1	72009	82	82
2	1406?	37	37
3	1702	1	1
4	1145	5	5
5			197
6			61
7	3017	55	54
8	5401	51	51
9	2822	61	61
10	715?	54	54
11	4375	57	57
12	2350	1	1
13	240?	1	1
14	111?	1	1

Compiler after Reorganization of procedures :

The following procedures have been grouped together:
endofline, nextch, insymbol, enterid, searchid,
putic, genl, selector, write, call, factor, term;
simpleexpression, expression, statement and body.

S I M U L A T I O N R E S U L T S
****

Total Number of TRAPS: 596

Number of Dummy TRAPS: 219

Number of Swap TRAPS: 377

Segment No.	Access Count	SwapinCount	SwapoutCount
0	140455	22	21
1	7781	36	36
2	2696	36	36
3	4094	56	55
4	13039	48	48
5	1730	39	39
6	640	2	2
7	1585	6	6
8	6129	43	42
9	1952	34	34
10	3003	38	38
11	6675	13	12
12	2350	1	1

10

1
1

1
2

1806
116

13
14

APPENDIX - 3
****

!PAS31b System Errors
****

1 .. write attempted on a write protected disk
2 .. seek/search error ; non data transfer error
3 .. bus grant late
4 .. circc error/read or write disable
5 .. synchronous error
6 .. drive unsafe
7 .. command check
10 .. sector over run
11 .. track over run
12 .. non existant memory
13 .. other errors
14 .. attempt to get/read beyond EOF
15 .. buffers exhausted cannot open file
16 .. file has not been opened for write
17 .. input data erroneous
20 .. too many digits for an integer

!P-Interpreter Error Messages
****

0 .. heap overflows stack
1 .. illegal instruction
2 .. not implemented
3 .. error in case statement
4 .. stack overflows heap
5 .. value out of range

!P-Assembler Error Messages
****

0 .. code array overflow
1 .. label table overflow
2 .. duplicated label
3 .. illegal instruction
4 .. unimplemented instruction

```
!P-Compiler Error Messages
*****************************
 1: error in simple type
 2: identifier expected
 3: 'program' expected
 4: ')' expected
 5: ':' expected
 6: illegal symbol
 7: error in parameter list
 8: 'of' expected
 9: '(' expected
10: error in type
11: '[' expected
12: ']' expected
13: 'end' expected
14: ';' expected
15: integer expected
16: '=' expected
17: 'begin' expected
18: error in declaration part
19: error in field-list
20: ',' expected
21: '*' expected

50: error in constant
51: ':=' expected
52: 'then' expected
53: 'until' expected
54: 'do' expected
55: 'to'/'downto' expected
56: 'if' expected
57: 'file' expected
58: error in factor
59: error in variable

101: identifier declared twice
102: low bound exceeds high bound
103: identifier is not of appropriate class
104: identifier not declared
105: sign not allowed
106: number expected
107: incompatible subrange types
108: file not allowed here
109: type must not be real
110: tagfield type must be scalar or subrange
111: incompatible with tagfield type
112: index type must not be real
113: index type must be scalar or subrange
114: base type must not be real
115: base type must be scalar or subrange
116: error in type of standard procedure parameter
```

13

```
117:: unsatisfied forward reference
118:: forward reference type identifier in variable declaration
119:: forward declared; repetition of parameter list not allowed
120:: function result type must be scalar, subrange or pointer
121:: file value parameter not allowed
122:: forward declared function; repetition of result type not allowed
123:: missing result type in function declaration
124:: F-format for real only
125:: error in type of standard function parameter
126:: number of parameters does not agree with declaration
127:: illegal parameter substitution
128:: result type of parameter function does not agree with declaration
129:: type conflict of operands
130:: expression is not of set type
131:: tests on equality allowed only
132:: strict inclusion not allowed
133:: file comparison not allowed
134:: illegal type of operand(s)
135:: type of operand must be boolean
136:: set element type must be scalar or subrange
137:: set element types not compatible
138:: type of variable is not array
139:: index type is not compatible with declaration
140:: type of variable is not record
141:: type of variable must be file or pointer
142:: illegal parameter substitution
143:: illegal type of loop control variable
144:: illegal type of expression
145:: type conflict
146:: assignment of files not allowed
147:: label type incompatible with selecting expression
148:: subrange bounds must be scalar
149:: index type must not be integer
150:: assignment to standard function is not allowed
151:: assignment to formal function is not allowed
152:: no such field in this record
153:: type error in read
154:: actual parameter must be a variable
155:: control variable must neither be formal nor nonlocal
156:: multidefined case label
157:: too many cases in case statement
158:: missing corresponding variant declaration
159:: real or string tag fields not allowed
160:: previous declaration was not forward
161:: again forward declared
162:: parameter size must be constant
163:: missing variant in declaration
164:: substitution of standard proc./func. not allowed
165:: multidefined label
166:: multideclared label
167:: undeclared label
168:: undefined label
169:: error in base set
170:: value parameter expected
```

```
                      APPENDIX - 4
           *************************
; Syntax of the P-machine's Assembly Language in BNF.
           *************************

<assembly program>  ::= <assembly record> <assembly record>
                        @ see note i
<assembly record>  ::= <assembly segment> END_OF_LINE
<assembly segment> ::= <assembly statement> END_OF_LINE | LAMBDA
<assembly statement> ::= I<integer> PPC<integer>|
                         I<integer>
                         @ see note ii
                         L<integer>|L<integer> "=" <integer>|
                         "<assembly instruction>

<relational instruction>  ::= <opcode class 5><char 1>|
                              <opcode class 5><integer>|
                              <opcode class 5> M<integer>

<procedure call instruction>  ::= CSP<standard procedure mnemonic>|
                                  CUP <integer> L<integer>

<return instruction> ::= RET<return what>
<return what> ::= P|R|I|B|C|A
<load constants instruction> ::= <load string> | <load others>
<load string> ::= LCA<string>
<load others> ::= LDC<load what>
<load what> ::= I<integer>|R<real>|N|
                B<integer>|N|

<integer list> ::= I<integer> | LAMBDA
<integer list> ::= "("<integer>")"
<ent instruction> ::= ENT<integer> L<integer>
<assembly instruction> ::=
                <opcode class 2><integer>|
                <opcode class 3><integer><integer>|
                <opcode class 4><integer>|
                <opcode class C <integer>|
                <opcode class 6><char 1>|
                <opcode class 7><char 1><integer>|
                <opcode class 8><char 1><integer><integer>|
                <relational instruction>|
                <procedure call instruction>|
                <return instruction>|
                <load constants instruction>|
                <ent instruction>|

<opcode class 1>  ::= ABI|ABR|ADI|ADR|AND|DEC|DIF|DVI|DVR|EOF|
                      FLT|FLO|INN|IOR|INC|INT|MOD|MPI|MPR|NGI|
                      NOT|ODD|SBI|SBR|SGS|SQI|SQR|UJC|STP|TRC|UNI

<opcode class 2>  ::= IXA|LAO|LAL|MOV|MST
<opcode class 3>  ::= LDA
<opcode class 4>  ::= UJP|XJP|FJP
<opcode class 5>  ::= EQU|NEQ|GEQ|LEQ|GRT|LES
<opcode class 6>  ::= STO
```

```
<opcode class 7> ::= LDI|LDO|STL|STO|IND
<opcode class 8> ::= LOD|STR|CHK
<char> ::= A|I|R|S
<standard procedure mnemonic> ::= GET|PUT|ELN|NEW|WRS|WLN|WRI|
                                  WRR|WRC|WRO|RDI|RDR|RDC|RST|
                                  SAV|STN|COS|EXP|LOG|SQT|ATN|RLN

<integer>,<string>,<real> and END_OF_LINE are as defined in Pascal.
The terminal symbols are either enclosed in quotes (Ex: "=") or
written in Upper case (Ex: T).

Note:
(i) The first assembly record of the assembly program consists of
    the whole Pascal program and should be loaded at PCODE[7] of
    the P-Machine. The second record to be loaded at PCODE[0],
    consists of a call to the outermost block of the pascal program.
(ii)<integer>,PPC<integer> indicates the number of the instruction to be
    generated. PPC<integer> indicates the value of the P-Machine's
    Program Counter at which the next instruction should be
    assembled. This statement is generated for every tenth instruction
    by the Compiler, to allow the reader of the code to relate it with
    the source program listing produced by the compiler.
```

APPENDIX - 5
****

:P-Machine's Instruction Set.
****

CODE	MNEMONIC	PARAMETERS	DESCRIPTION
0	ELN		Checks the EOLN condition for the file specified on the top of the stack;the result of this check is left on the top of the stack.
1	GET		performs GET on the file specified by the top of the stack and appropriately fills the buffer associated with it.
2	RDC		Reads a character from the file specified on the top of the stack and assigns it to the variable whose address is below the top of the stack;note the automatic updating of the buffer associated with the file.
3	RDI		Reads an integer from the file specified on the top of the stack and assigns it to a variable whose address is below the top of the file;note the automatic updating of the buffer associated with the file.
4	RDR		Reads a real number from the file specified on the top of the stack and assigns it to a variable whose address is below the top of the stack;note the automatic updating of the buffer associated with the file.
5	RLN		The top of the stack specifies a file on which a READLN is performed; note the automatic updating of the buffer associated with the file.
6	PUT		Performs the PUT operation on the file specified by the top of the stack.
7	WRC		Writes on the file specified on the top of the stack a character which is found immediately below the top of the stack. Just below the top is the number of characters to be written out.
8	WRI		Writes on the file specified by the top of the stack an integer whose value is given immediately below the element below the top of the stack. Just below the top is the number of characters to be written out.
9	WRO		Same as WRI, but Octal digits are written out.
10	WRR		Writes on the file specified by the top of the stack a real number whose value is given immediately below the element below the top of the stack. Just below the top is the number of characters to be written out.
11	WRS		Writes on the file specified by the top of the stack a string of characters; immediately below the top of the stack is the value of the actual length of the string;below this is specified before WRS is executed the string to be written must be loaded into the P-Machine's array STRING.

No.	Mnemonic	Description
12	WLN	performs WRITELN on the file specified on the top of the stack.
13	NEW	The top of the stack specifies the size of element to be allocated from the free storage; the address of the element is to be stored in the pointer variable whose address is to be found below the top of the stack.
14	PAK	Sets the 'NEW' pointer (heap pointer) to the pointer value on the top of the stack.
15	RST	
16	SAV	Saves the current value of the 'NEW' pointer (heap pointer) at the address specified on the top of the stack.
17	SIN	Top of the stack := SINE(top of the stack).
18	CUS	Top of the stack := COSINE(top of the stack).
19	ATN	Top of the stack := ARCTANGENT(top of the stack).
20	EXP	Top of the stack := e ^ top of the stack.
21	LOG	Top of the stack := Natural Logarithm of top of the stack.
22	SQT	Top of the stack := Square Root of top of the stack.
32	EOF	test on end of file
33	NOT	Boolean NOT
34	IOR	Boolean 'inclusive or'
35	AND	Boolean AND
36	LODCBF	Load Constant Boolean FALSE
37	LODCBT	Load Constant Boolean TRUE
38	LESB	compare Boolean less than
39	LEOB	compare Boolean less than or equal
40	EQOB	compare boolean equal
41	NEOB	compare Boolean not equal
42	GEOB	compare Boolean greater than or equal
43	GRTB	compare Boolean greater than
44	ODD	test on odd
45	NGII	integer sign inversion
46	ADII	integer addition
47	SBII	integer subtraction
48	MPII	integer multiplication
49	DVII	integer division
50	MOD	modulus
51	SQI	square integer
52	ABI	absolute value of integer
53	LODCN	load constant NIL
54	LESA	compare address less than
54	LESI	compare integer less than
55	LEOA	compare address less than or equal
55	LEOI	compare integer less than or equal
56	EQUA	compare address equal
56	EQUI	compare integer equal
57	NEOA	compare address not equal
57	NEOI	compare integer not equal
58	GEOA	compare address greater than or equal
58	GEOI	compare integer greater than or equal
59	GRTA	compare address greater than
59	GRTI	compare integer greater than
60	SGS	generate singleton set
61	INN	test set membership (IN)

No.	Mnemonic	Param	Description
62	UNI		set union
63	INT		set intersection
64	DIF		set difference
65	LESS		compare Set less than
66	LEQS		compare Set less than or equal
67	EQOS		compare Set equal
68	NEQS		compare Set not equal
69	GEQS		compare Set greater than or equal
70	GRTS		compare Set greater than
71	FTRC		truncation
72	FLT		float next to the top
73	FLT		float top of the stack
74	NGR		real sign inversion
75	ADR		real addition
76	SBR		real subtraction
77	MPR		real multiplication
78	DVR		real division
79	SQR		square real
80	ABR		absolute value of real number
81	LESR		compare Real less than
82	LEQR		compare Real less than or equal
83	EQUR		compare Real equal
84	NEQR		compare Real not equal
85	GEQR		compare Real greater than or equal
86	GRTR		compare Real greater than
87	UJC		error in case statement
88	STP		stop
133	LESM	O	compare multiple units less than
134	LEQM	O	compare multiple units less than or equal
135	EQOM	O	compare multiple units equal
136	NEQM	O	compare multiple units not equal
137	GEQM	O	compare multiple units greater than or equal
138	GRTM	O	compare multiple units greater than
139	DEC	O	decrement
140	INC	O	increment
141	IXA	O	compute indexed address
142	LAL	O	load current-level address
143	LAO	O	load base-level address
144	FJP	O	false jump
145	XJP	O	indexed jump
146	UJP	O	unconditional jump
147	ENT	O	enter block
148	MOV	O	move
149	LDCI	O	load constant integer
150	LDA	P	load address with level P
151	CUP	P	call user procedure
152	MST	P	mark stack
153	RET	P	return from block
154	CHK	L	check against upper and lower bounds

No.	Mnemonic	STRING	Param	Description
155	LCA	STRING	0	load string, the string is terminated by a 0
192	LODK	R		load constant real
193	LODS	S		load constant set
194	INDA		0	indexed fetch address
194	INDI		0	indexed fetch integer
194	INDB		0	indexed fetch boolean
195	INDR		0	indexed fetch real
196	INDS		0	indexed fetch set
197	LODA		0	load contents of current-level address
197	LODI		0	load contents of current-level integer
198	LODB		0	load contents of current-level boolean
198	LODR		0	load contents of current-level real
199	LODS		0	load contents of current-level set
200	LODA		0	load contents of base-level address
200	LODI		0	load contents of base-level integer
201	LODB		0	load contents of base-level boolean
202	LODR		0	load contents of base-level real
203	LODS		0	load contents of base-level set
203	LODA	P	0	load contents of address address
204	LODI	P	0	load contents of address integer
205	LODB	P	0	load contents of address boolean
206	LODR	P	0	load contents of address real
206	LODS	P	0	load contents of address set
207	SIDA		0	store indirect address
208	SICI		0	store indirect integer
208	SIDB		0	store indirect boolean
209	SIOR		0	store indirect real
209	SIOS		0	store indirect set
210	STLA		0	store at current-level address
211	STLI		0	store at current-level integer
212	STLB		0	store at current-level boolean
212	STLR		0	store at current-level real
212	STLS		0	store at current-level set
213	SROA		0	store at base-level address
214	SROI		0	store at base-level integer
215	SROB		0	store at base-level boolean
215	SROR		0	store at base-level real
215	SROS		0	store at base-level set
216	STRA	P		store at address address
217	STRI	P		store at address integer
	STRB	P		store at address boolean
	STRR	P		store at address real
	STRS	P		store at address set

Note:
(i) * : if necessary leading blanks are filled in.
    P : Parameter Section
(ii) _ : one byte
     U : two bytes
     O : two bytes
     R : four bytes(real)
     S : eight bytes(set)

APPENDIX - 6
****

```
*****  ********************************  *****
*****  *                                *  *****
*****  *                                *  *****
*****  *            PAS316              *  *****
*****  *                                *  *****
*****  *   A SINGLE USER PASCAL         *  *****
*****  *   PROGRAMMING STATION          *  *****
*****  *                                *  *****
*****  ********************************  *****
```

```
%**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************
****                                                              ****
****   MODULE : PAS316                                            ****
****   DATE   : 1 JULY 1982                                       ****
****   AUTHOR : S SRINIVAS RAGHURAM                               ****
****   MAIN MODULE : Command Processor                            ****
****   VALID COMMANDS :                                           ****
****     ASMINT ...: Assemble and Interpret                       ****
****     COPX   ...: Copy ASCII files                             ****
****     PASASM ...: Compile and Assemble                         ****
****     PASCAL ...: Compile                                      ****
****     PASGO  ...: Compile, Assemble and Interpret              ****
****     ASM    ...: Assemble                                     ****
****     EX     ...: Interpret                                    ****
****     PRINT  ...: Print file on line printer                   ****
****     TYPE   ...: Type file on tty                             ****
****     KJOB   ...: Exit from PAS316                             ****
****                                                              ****
**********************************************************************
**********************************************************************
**********************************************************************
**********************************************************************

MODULE pas316( MAIN , STACK , LIST )=
BEGIN

MACRO lsby = 0,8$;

BIND cmptrk = 33, cmpsrfc = 0, cmpsectr =1;
   ! Compiler's P=CODE disk file address
BIND false = 0, true = 1,
   space =#40, slash =#57, ave = #101, yell =#114, owe = #117,
   yes =#123, tee =#124, zed = #132, save =#141,
   szed =#172, crr = #15, lf = #12, ctrlu = #25, star = #52,
   stkptr =#177742, cmdtabsize =10;
BIND version = UPLIT ASCIZ "PAS316 Pascal System Version of Jul 1,82",
   compver = UPLIT ASCIZ "PAS316 P=Compiler Version of Jul 1,82",
   mes1 = UPLIT ASCIZ "Give Object",
   mes2 = UPLIT ASCIZ "Illegal Command",
   mes3 = UPLIT ASCIZ "Give Source",
   mes4 = UPLIT ASCIZ "Give Destn";

STRUCTURE cmmdtable[ i ] =
(.cmmdtable + .i * 6);

BIND cmmdtable cmdtab = UPLIT(
   "COPY" ,   "PASASM" , "PASCAL" , "PASGO" ,
   "ASMINT" , "PRINT" , "TYPE" , "KJOB" = );
   "ASM" ,    "EX" ,

GLOBAL savstkval;
GLOBAL ttyno;

BYTE OWN VECTOR IDI 6 ];
OWN ch , k , t , c , s;
OWN list , trace , obj , ttyout , abrt , notquit;
```

```
EXTERNAL getlch,ttcrlf,ttoutstr,putlch;
EXTERNAL ttinit,diskinit,inittraps,initmap,initbufs;
EXTERNAL asm,intr,psave,pload;
EXTERNAL getfilename,copy,type;

ROUTINE getch =
BEGIN
    getlch(.ttyno,ch);
    IF .ch GEQ save THEN
        IF .ch LEQ sze THEN ch = .ch - #40;
    IF .ch LEQ cr THEN putlch(.ttyno,lf)
        ELSE IF .ch EOL lf THEN putlch(.ttyno,cr)
        ELSE IF .ch EOL ctrlu THEN abrt = true;
END; ! getch
END;

ROUTINE neqcomp =
BEGIN
    LOCAL c,b;
    c = 0; b = true;
    WHILE .b AND (.c LEQ 5) DO
        IF .idf.c]<lsby> EOL .(cmatabl.k]+.c)<lsby> THEN c = .c + 1
            ELSE b = false;
    RETURN not .b
END; ! neqcomp

ROUTINE setswitches =
BEGIN
    DO
    BEGIN
        getch();
        IF .ch EOL yell THEN list = NOT .list
            ELSE IF .ch EOL tee THEN trace = NOT .trace
            ELSE IF .ch EOL yes THEN obj = NOT .obj
            ELSE IF .ch EOL owe THEN ttyout = NOT .ttyout;
        getch()
    END
    UNTIL .ch NEQ slash
END; ! setswitches

ROUTINE compile =
BEGIN
    pload(cmptrk,cmpsrfc,cmpsectr);
    ttcrlf(.ttyno); ttoutstr(.ttyno,compver);
    intr(.trace)
END; ! compile

ROUTINE assemble =
BEGIN
    LOCAL n;
    n = asm(.list);
    IF .obj THEN
    BEGIN
        ttcrlf(.ttyno); ttoutstr(.ttyno,mes1);
        getfilename(t,c,s);
```

```
        psave(.t,.c,.s,.n)
    END; ! assemble

GLOBAL ROUTINE getcommand =
    BEGIN
    ttcrlf(.ttyno);
    putlch(.ttyno,star); ch = space;
    list = trace = obj = abrt = false; ttyout = true;
    INCR i FROM 0 TO 5 DO id[.i] = .ch;
    WHILE .ch GEQ space DO getch();
    IF .ch GEQ ave AND .ch LEQ zed THEN
        BEGIN
        LOCAL ind;
        ind = 0;
        WHILE .ch GEQ ave AND .ch LEQ zed DO
            BEGIN
            IF .ind LEQ 5 THEN id[.ind] = .ch;
            ind = .ind + 1;
            getch()
        END;
    INCR i FROM 0 TO 5 DO
        (cmdtab[0] + .i)<lsby> = .id[.i];
        k = cmdtabsize;
    WHILE neqcomp() DO k = .k -1;
    IF .ch EQL slash THEN setswitches();
    IF .abrt THEN RETURN;
    CASE .k OF
    SET
    =0:
        BEGIN
        ttcrlf(.ttyno); ttoutstr(.ttyno,mes2)
        END;
    =1 ASMINT
        BEGIN
        assemble(); intr(.trace,.ttvout)
        END;
    =2 COPY
        BEGIN
        LOCAL t1,c1,s1;
        ttcrlf(.ttyno); ttoutstr(.ttyno,mes3);
        getfilename(t,c,s);
        ttcrlf(.ttyno); ttoutstr(.ttyno,mes4);
        getfilename(t1,c1,s1);
        copy(.t,.c,.s,.t1,.c1,.s1)
        END;
    =3 PASASM
        BEGIN
        compile(); obj = NOT .obj; assemble()
        END;
```

01090
01100
01110
01120
01130
01140
01150
01160
01170
01180
01190
01200
01210
01220
01230
01240
01250
01260
01270
01280
01290
01300
01310
01320
01330
01340
01350
01360
01370
01380
01390
01400
01410
01420
01430
01440
01450
01460
01470
01480
01490
01500
01510
01520
01530
01540
01550
01560
01570
01580
01590
01600
01610
01620

```
:4 PASCAL
  compile();

:5 PASGO
BEGIN
  compile(); assemble(); intr(.trace,.ttvout)
END;

:6 ASM
BEGIN
  obj = NOT .obj; assemble()
END;

:7 EX
BEGIN
  ttcrlf(.ttyno); ttoutstr(.ttyno,mes1);
  getfilename(t,c,s);
  pload(.t,.c,.s);
  intr(.trace,.ttvout)
END;

:8 PRINT
;

:9 TYPE
BEGIN
  ttcrlf(.ttyno); getfilename(t,c,s); type(.t,.c,.s)
END;

:10 KJOB
  notquit = false

TES;
END ! IF
END; : getcommand

: MAIN program of PAS316
reset();
ttyno = 4;
initmap();
inittraps();
ttinit();
diskinit();
initbufs();
ttcrlf(.ttyno); ttoutstr(.ttyno,version);
notquit = true;
savstkval = .stkptr - 2;
WHILE .notquit DO getcommand() ;

END
ELUDOM
```

```
01630
01640
01650
01660
01670
01680
01690
01700
01710
01720
01730
01740
01750
01760
01770
01780
01790
01800
01810
01820
01830
01840
01850
01860
01870
01880
01890
01900
01910
01920
01930
01940
01950
01960
01970
01980
01990
02000
02010
02020
02030
02040
02050
02060
02070
02080
02090
02100
02120
```

```
****************************************************
****************************************************
****                                            ****
****   MODULE  : PASSEMBLER                      ****
****   DATE    : 1 JULY 1982                     ****
****   AUTHOR  : S. SRINIVAS RAGHURAM            ****
****   INPUT   : P-Machine's Assembly Language Program
****                in file PRR.                 ****
****   OUTPUT  : The object code is loaded into the
****                P-Machine's-code array PCODE.****
****   LSTFIL  : Listing is optionally produced. ****
****   ENTRY   : ASM(LSTN);                      ****
****                lstn = true then assembly listing
****                is generated.                ****
****   RETURNS: Maximum value of Location Counter PPC
****                                            ****
****************************************************
MODULE passembler( unsafe , nooptimize ,nofinal , nolist ) =

BEGIN
% The P-Assembler has almost no error detection or error recovery.
  It expects a clean and correct input , typically produced by a
  Compiler.
%

        Assembler ERRORS:
        0.. CODE ARRAY OVERFLOW
        1.. LABEL TABLE OVERFLOW
        2.. DUPLICATED LABEL
        3.. ILLEGAL INSTRUCTION
        4.. UNIMPLEMENTED INSTRUCTION
%

MACRO
  lsby = 0,8 $;   !: least significant byte
  msby = 8,8 $;   !: most significant byte

BIND
! ASCII characters
aye = #101, bee = #102, sea = #103, eye = #111, vell = #114,
emm = #115, ven = #116, pee = #120, are = #122, ves = #123,
lpar = #50 , rpar = #51 , quote= #47 , space= #40 , nul = #0 ,
zero= #60 , nine = #71 , plus = #53 , minus= #55 , equal= #75 ;

! P-Machine OpCodes
LDCIP =#149, LDCRP = 192, LDCNP =#53 , LDCBFP= 36, LDCBTP= 37,
LDCSP =#193, CSPP == 0,155, CHKP =#154, LDCP =#36, CUPP ==151,
RETP =#153, LCAP =#194, INDP ==194, LDLP =#197, LDOP ==200,
STLP =#209, SKOP =#212, LOOP ==212, STRP ==215, LESP = 38,
LEOP =#209, EQUP = 40, NEOP = 40, GEQP =#41, GRTP = 43,
DECP =#139, INCP =#140, IXAP =#140, LALP =#141, LAOP ==143,
MOVP =#148, FJPP =#148, XJPP =#144, UJPP =#145, ENTP ==147,
STOP =#206, LDAP. =#150, MSTP =#152;

! Constants
```

```
      w1 = 0, w2 = 1, false = 0, true = 1, entered = 0, defined = 1,
      nil = -1, nosis = 61, !number of instructions = 1
      nosip = 22, !number of special functions/procedures - 1
      codemax = #40000, !size of P-Machine's CODE SEGMENT
      maxlabel = 1999, !maximum number of labels assembler can handle
      begincode = 7, !start of code of segment1, segment2 starts at 0
      dirr = 3, !input file for assembler
      lstil = 2; !listfile produced by assembler

BIND
      version = UPLIT ASCIZ "PAS316 P-Assembler Version of Jul 1,82",
      mes1    = UPLIT ASCIZ "End of Assembly"
      err0    = UPLIT ASCIZ "CODE ARRAY OVERFLOW",
      err1    = UPLIT ASCIZ "LABEL TABLE OVERFLOW",
      err2    = UPLIT ASCIZ "DUPLICATED LABEL",
      err3    = UPLIT ASCIZ "ILLEGAL INSTRUCTION",
      err4    = UPLIT ASCIZ "UNIMPLEMENTED INSTRUCTION";

% to understand the expansion of the following 2 MACRO's refer
% bliss-11 Programmer's Manual bp52
MACRO
      initopcodes( aryname )[] =
            loadopcode( aryname ; $REMAINING ) $;
MACRO
      loadopcode( start )[opval ] =
            start[SCOUNT] = opval $;

STRUCTURE insttable[ i , field ] =
      ( .insttable + .i * 4 + .field * 2 );

BIND insttable inst = UPLIT(
      "ABI" = , "ADI" = , "ANR" = , "AND" = , "CHK" = , "CSP" = , "CUP" = ,
      "DEC" = , "DIF" = , "DVR" = , "ENT" = , "EOF" = , "EOD" = , "FJP" = ,
      "FLO" = , "FLT" = , "GRT" = , "INC" = , "IND" = , "INT" = ,
      "IOR" = , "IXA" = , "LAO" = , "LDA" = , "LDC" = , "LDL" = ,
      "LOD" = , "LEQ" = , "LOD" = , "MOD" = , "MOV" = , "MPI" = , "MPR" = ,
      "MST" = , "NGI" = , "NGR" = , "MOD" = , "ODD" = , "RET" = , "SRI" = ,
      "SBR" = , "SGS" = , "SOR" = , "SRO" = , "STL" = , "STO" = , "STP" = ,
      "STR" = , "TRC" = , "UJC" = , "UJP" = , "UNI" = , "XJP" = );

BIND insttable sptab = UPLIT(
      "ELN" = , "GET" = , "RDC" = , "RDR" = , "RDI" = , "RLN" = , "PUT" = , "WRC" = ,
      "WRI" = , "WRO" = , "WRR" = , "WRS" = , "WLN" = , "NEW" = , "PAK" = , "RST" = ,
      "SAV" = , "SIN" = , "COS" = , "ATN" = , "EXP" = , "LOG" = , "SOT" = );

EXTERNAL endofile , endofline , fileclose;
EXTERNAL ttoutstr, ttputoct, ttcrlf, putich;
EXTERNAL read , write , readin , writein , readint , reset , rewrite;
EXTERNAL getfilename , deposit , fetch;
EXTERNAL ttyno;

OWN ppc ,    ! P-Machine's Program Counter
```

```
op    ,    ;opcode
op    ,    ;p register
q     ,    ;p register
ch    ,    ;index variable used for searching tables
           ;look ahead character for file PRR
lablevalue;  ;label value for updating labeltable

OWN  prradr;  ! buffer for file PRR

OWN  lstadr;  ! buffer for file LSTFIL

OWN  listing , prnt;

BYTE OWN VECTOR opc[ nols ];

BYTE OWN VECTOR lblst[ maxlabel + 1 ];

OWN VECTOR lblval[ maxlabel + 1 ];

OWN VECTOR ld[ 2 ];

ROUTINE initassembler =
BEGIN
LOCAL trk,srfc,sectr;
initopcodes( OPC
52 ,80 ,46 ,75 ,35 ,154, 0 ,151,139,64  ,49 ,78 ,147,32 ,40 ,
144 ,72 ,73 ,42 ,43 ,140,194,61 ,63 ,34  ,141,142,143,155,150,
36 ,197,200,39 ,38 ,203,59 ,148,48 ,77  ,152,41 ,45 ,74 ,33 ,
44 ,153,47 ,76 ,60 ,51 ,79 ,212,209,206,88 ,215,71 ,87 ,146, );

opc = begincode;
print = true;
ttcrlf(.ttyno); getfilename(trk,srfc,sectr);
reset(prr,prradr,.trk,.srfc,.sectr);
IF listing THEN
BEGIN
ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "Listing");
getfilename(trk,srfc,sectr);
rewrite(lstfil,lstadr,.trk,.srfc,.sectr);
END;
ld[ w1 ]<lsby> = ld[ w1 ]<msby> = ld[ w2 ]<lsby> = ld[ w2 ]<msby> = space;
INCR i FROM 0 TO maxlabel DO
BEGIN
lblval[ .i ] = nil; lblst[ .i ] = entered;
END;
END;  ! initassembler

ROUTINE asserr( n )=
BEGIN
LOCAL errmesg;
ttcrlf(.ttyno);
ttoutstr(.ttyno,UPLIT ASCIZ "Assembly Error: ");
ttputoct(.ttyno,.n);
```

```
016300      ttcrlf(.ttyno);
016400    CASE .n OF
016500    SET
016600      errmesg = err0;
016700      errmesg = err1;
016800      errmesg = err2;
016900      errmesg = err3;
017000      errmesg = err4
017100      TES;
017200    ttoutstr(.ttyno,.errmesg); ttcrlf(.ttyno);
017300    EMT(0);
017400    END; ! asserr
017500
017600    MACRO octch(f,dig) = write(f,zero+dig)s;
017700
017800    ROUTINE writbyte(f,value)=
017900    BEGIN
018000    LOCAL dig;
018100      dig = .value<6,2>; octch(.f,.dig);
018200      dig = .value<3,3>; octch(.f,.dig);
018300      dig = .value<0,3>; octch(.f,.dig)
018400    END; ! writbyte
018500
018600    ROUTINE writword(f,value)=
018700    BEGIN
018800    LOCAL dig;
018900      dig = .value<15,1>;octch(.f,.dig);
019000      INCR i FROM 1 TO 5 DO
019100      BEGIN
019200      dig = .value<12,3>; octch(.f,.dig);
019300      value = .value*8
019400      END
019500    END; ! writword
019600
019700    ROUTINE putword( adr , value )=
019800    BEGIN
019900    LOCAL tmp;
020000      tmp = .value<lsby>; deposit(.adr,.tmp);
020010      tmp = .value<msby>; deposit(.adr+1,.tmp)
020020    END; !putword
020030
020040    ROUTINE getword( adr )=
020050    BEGIN
020060    LOCAL value;
020070      value<lsby> = fetch( .adr );
020080      value<msby> = fetch( .adr + 1 );
020090      RETURN .value;
020100    END; ! getword
020110
020120    ROUTINE loadbyte( value )=
020130    BEGIN
020140    IF .listing THEN
020150    IF .print THEN
020160      BEGIN
```

```
02170          INCR i FROM 0 TO 3 DO
02180             write(lstfil,space);
02190             writbyte(lstfil,.value)
02200      !&debug; putlch(.ttyno,space);    ttputoct(.ttyno,.value);  putlch(.ttyno,space);
02210          END;
02220          deposit(.ppc,.value);
02230          ppc = .ppc + 1;
02240          IF .ppc GTR codemax THEN asserr(0);
02250          END; !.loadbyte
02260
02270      ROUTINE loadword( value ) =
02280      BEGIN
02290          LOCAL val;
02300          IF .listing THEN
02310          BEGIN
02320             write(lstfil,space);
02330             writword(lstfil,.value);
02340      !&debug; putlch(.ttyno,space); ttputoct(.ttyno,.value);
02350             prnt = false
02360          END;
02370          val = .value<lsby>;
02380          loadbyte(.val );
02390          val = .value<msby>;
02400          loadbyte(.val );
02410          IF .listing THEN  prnt=true;
02420          END; !.loadword
02430
02440      ROUTINE update( x ) =
02450      !when a label definition LX is found
02460      BEGIN
02470          LOCAL curr, succ, endlist;
02480          IF .x GTR maxlabel THEN asserr(1);
02490          IF .lblist[ .x ] EQL defined THEN asserr( 2 )
02500          ELSE
02510          BEGIN
02520             IF .lblval[ .x ] NEQ nil THEN
02530             BEGIN
02540                curr = .lblval[ .x ]; endlist = false;
02550                UNTIL .endlist DO
02560                BEGIN
02570                   succ = getword( .curr );
02580                   putword( .curr, .lablevalue );
02590                   IF .succ EQL nil THEN endlist = true
02600                   ELSE curr = .succ;
02610                END
02620             END;
02630             lblist[ .x ] = defined;
02640             lblval[ .x ] = .lablevalue;
02650          END;
02660          END; ! update
02670
02680      ROUTINE lookup( x ) =
02690      !search in label table
02700
```

```
02710  BEGIN
02720  IF .x GTR maxlabel THEN asserr(1);
02730  IF .lblst[ .x ] EOL entered THEN
02740      BEGIN
02750      lblval[ .x ] = .ppc;
02760      loadword(nil);
02770      END
02780  ELSE
02790      BEGIN
02800      LOCAL temp;
02810      temp = .ppc;
02820      loadword(.lblval .x );
02830      lblval[ .x ] = .temp;
02840      END
02850  ELSE ! .lblst[ .x ] EOL defined
02860      loadword(.lblval .x ] );
02870  END; ! lookup
02880
02890
02900  ROUTINE labelsearch =
02910  BEGIN
02920  LOCAL x;
02930  WHILE .ch NEQ yell AND NOT endofline(prr) DO
02940      read( prr, ch );
02950  readint(prr, x);
02960  lookup( .x );
02970  END; ! labelsearch
02980
02990  FORWARD assembler;
03000
03010  ROUTINE generate =
03020  ! generate segment of code
03030  BEGIN
03040  LOCAL x; ! label number
03050  UNTIL endofline( prr ) DO
03060      BEGIN
03070      read( prr, ch );
03080      IF .ch EOL eye THEN readin( prr )
03090      ELSE
03100      IF .ch EOL yell THEN
03110          BEGIN
03120          readint( prr, x );
03130          IF NOT endofline( prr ) THEN read( prr, ch );
03140          IF .ch EOL equal THEN readint( prr , lablevalue )
03150          ELSE lablevalue = .ppc;
03160          update( .x ); readin( prr );
03170          END
03180      ELSE ! .ch EOL space
03190          BEGIN
03200          DO read( prr, ch ) UNTIL .ch NEO space;
03210          IF listing THEN
03220              BEGIN
03230              writeln(lstfil); writword(lstfil,.ppc);
03240              &debug% ttcrlf(.ttyno); ttputoct(.ttyno,.ppc); putch(.ttyno,space);
```

```
        write(lstfil,quote)
      END;
    assembler();
      END
    END
  END; ! until
END; ! generate

ROUTINE assembler =
BEGIN

ROUTINE getid =
!-!check <msby> & <lsby> are correct in this context
BEGIN
LOCAL c, j<lsbv>;
id[w1 j<lsbv> = .ch;
read( prr , c ); id[ w1 ]<msby> = .c;
read( prr , c ); id[ w2 ]<lsby> = .c;
!-!idf w2 j<msby> is SPACE always
IF listing THEN
BEGIN
INCR i FROM 0 TO 3 DO
write(lstfil,space);
c = .id[w1]<lsby>;
write(lstfil,.c);
%debug% putich(.ttyno,.c);
c = .id[w1]<msby>;
write(lstfil,.c);
%debug% putich(.ttyno,.c);
c = .id[w2]<lsby>;
write(lstfil,.c);
%debug% putich(.ttyno,.c)
END;
IF NOT endofline( prr ) THEN
BEGIN
read( prr ,ch);
%debug% putich(.ttyno,.ch);
IF .listing THEN write(lstfil,.ch)
END;
END; ! getid

MACRO maccomp(oper,table)=
.(id[w1]+.q)<0,8> OPER .(table[.k,w1]+.q)<0,8> $;

ROUTINE compare=
BEGIN
q = 0; p = true;
WHILE .p AND (.q LEQ 3) DO
IF maccomp(EQL,inst) THEN q = .q + 1
ELSE p = false;
END; ! compare

ROUTINE legcomp=
BEGIN
compare();
```

03250
03260
03270
03280
03290
03300
03310
03320
03330
03340
03350
03360
03370
03380
03390
03400
03410
03420
03430
03440
03450
03460
03470
03480
03490
03500
03510
03520
03530
03540
03550
03560
03570
03580
03590
03600
03610
03620
03630
03640
03650
03660
03670
03680
03690
03700
03710
03720
03730
03740
03750
03760
03770
03780

```
037900        RETURN (( maccomp(LEU,inst)) OR .p )
038000      END; ! leqcomp
038100
038200    ROUTINE geqcomp=
038300      BEGIN
038400      compare();
038500        RETURN (( maccomp(GEO,inst)) OR .p)
038600      END; ! geqcomp
038700
038800    ROUTINE neqcomp=
038900      BEGIN
039000      q = 0; p = true;
039100      WHILE .p AND (.q LEO 3 ) DO
039200      IF maccomp(EOL,sptab) THEN q = .q + 1
039300        ELSE p = false;
039400        RETURN NOT .p;
039500      END; ! neqcomp
039600
039700    ROUTINE binarysearch=
039800      BEGIN
039900      LOCAL i , j;
040000      i = 0; j = nois;
040100      DO
040200        BEGIN
040300        k = (.i + .j ) DIV 2;
040400 !&debug#putich(.ttyno,.inst[.k,w1]<lsby>);putich(.ttyno,.inst[.k,w1]<msby>);putich(.ttyno,.inst[.k,w2]<lsby>...
040500 !&debug#putich(.ttyno,.inst[.k,w2]<lsby>);putich(.ttyno,.inst[.k,w2]<msby>);ttcrlf(.ttyno);
040600        IF leqcomp() THEN j = .k - 1;
040700        IF geqcomp() THEN i = .k + 1;
040800        END
040900      UNTIL .i GTR .j;
041000      IF .i-1 GTR .j THEN op = .opc[ .k ]
041100        ELSE asser( 3 );
041200      END; ! binarysearch
041300
041400    ROUTINE relopinsts =
041500      BEGIN
041600      IF .ch EOL ave OR .ch EOL eye THEN k = .op + 16
041700      ELSEIF .ch EOL are THEN k = .op + 43
041800      ELSEIF .ch EOL yes THEN k = .op + 27
041900      ELSEIF .ch EOL emm THEN
042000        BEGIN
042100        k = .op + 95;
042200        readint( pri , q );
042300        END
042400      ELSE ! .ch EOL bee
042500        k = .op;
042600      loadbyte( .k );
042700      IF .ch EOL emm THEN loadword( .q );
042800      END; ! relopinsts
042900
043000    ROUTINE noadchinsts =
043100      BEGIN
043200      loadbyte( .op );
```

```
        readint( prr , q );
        loadword( .q );
END; !noadchinsts

ROUTINE entjumps =
BEGIN
    loadbyte( .op );
    labelsearch();
END; !entjumps

ROUTINE oneopldst =
BEGIN
    k := ( IF .ch EOL are THEN .op + 1
           ELSE IF .ch EOL yes THEN .op + 2
           ELSE .op %A,R,T% );
    loadbyte( .k );
    readint( prr , q );
    loadword( .q );
END; !oneopldst

ROUTINE twoopldst =
BEGIN
    k := ( IF .ch EOL are THEN .op + 1
           ELSE IF .ch EOL yes THEN .op + 2
           ELSE .op %A,R,T,space% );
    loadbyte( .k );
    readint( prr , q );
    readint( prr , q );
    loadbyte( .p );
    loadword( .q );
END; !twoopldst

! main program of assembler
getid();
binarysearch();
SELECT .op OF
NSET :
CSPP :  BEGIN
        WHILE .ch EOL, space DO read( prr, ch );
        getid(); k := 0; ! linear search of SPTABLE
        WHILE neqcomp() DO k := .k + 1;
        ! NOTE : no boundary checks
        loadbyte( .k );
        END;
CHKP :  BEGIN
        loadbyte( .op );
        INCR i FROM 0 TO 1 DO
        BEGIN
            readint( prr , q ); loadword( .q );
        END
        END;
LDCP :  BEGIN
        IF .ch EOL eve THEN
        BEGIN
```

```
04330
04340
04350
04360
04370
04380
04390
04400
04410
04420
04430
04440
04450
04460
04470
04480
04490
04500
04510
04520
04530
04540
04550
04560
04570
04580
04590
04600
04610
04620
04630
04640
04650
04660
04670
04680
04690
04710
04720
04730
04740
04750
04760
04770
04780
04790
04800
04810
04820
04830
04850
04860
```

```
        loadbyte( LDCIP );
        readint( prr , q );
        loadword( .q );
      END;
    ELSE IF .ch EOL are  THEN  asserr( 4 );
    ELSE IF .ch EOL yen  THEN  loadbyte( LDCNP ) := R
    ELSE IF .ch EOL bee  THEN       := N
      BEGIN
        readint( prr , p );
        loadbyte( IF .p THEN LDCBIP ELSE LDCBFP ));
      END;
    ELSE IF .ch EOL lpar THEN
      BEGIN
        LOCAL VECTOR setval[ 4 ];
        loadbyte( LDCSP );
        INCR i FROM 0 TO 3 DO setval[ .i ] = 0;
        read( prr , ch );
        WHILE .ch NEO rpar DO
          BEGIN
            readint( prr , q );
            setval[ .q DIV 16 ] = .setval[ .q DIV 16 ]
                  .OR ( 1 ^ ( .q MOD 16 ) );
            read( prr , ch );
          END;
        TNCR i FROM 0 TO 3 DO loadword( .setval[ .i ] );
      END; !s
  END; ! LDCP

CUPP :: BEGIN
          loadbyte( .op );
          readint( prr , p ); loadbyte( .p );
          labelsearch();
        END;

MSTP :: BEGIN
          loadbyte( .op );
          readint( prr , p );
          loadbyte( .p );
        END;

RETP :: BEGIN
          loadbyte( .op );
          p := (IF .ch EOL eve  THEN 1
                ELSE IF .ch EOL are  THEN 2
                ELSE IF .ch EOL sea  THEN 3
                ELSE IF .ch EOL bee  THEN 4
                ELSE IF .ch EOL ave  THEN 5
                ELSE IF .ch EOL pee  THEN 0 );
          loadbyte( .p );
        END; !RETP

LCAP :: BEGIN
          loadbyte( .op );
          read( prr , ch );
          ! .ch is the first character of the string
          WHILE .ch NEO quote DO
            BEGIN
              loadbyte( .ch );
```

04870
04880
04890
04900
04910
04920
04930
04940
04950
04960
04970
04980
04990
05000
05010
05020
05030
05040
05050
05060
05070
05080
05090
05100
05110
05120
05130
05140
05150
05160
05170
05180
05190
05200
05210
05220
05230
05240
05250
05260
05270
05280
05290
05300
05310
05320
05330
05340
05350
05360
05370
05380
05390
05400

```
          read( prr , ch );
        END;
        loadbyte( nul );
      END; !LCAP - Pinstruction
  STOP : BEGIN ! STO - Pinstruction
      K := ( IF .ch EOL are THEN .op + 1
             ELSE IF .ch EOL yes THEN .op + 2
             ELSE .op % A,B,I% );
        loadbyte( .k );
        END;
LESP     relopinsts();
LEOP     relopinsts();
EEOP     relopinsts();
NEOP     relopinsts();
GETP     relopinsts();
DECP     noadchinsts();
TNJP     noadchinsts();
TXJP     noadchinsts();
LALP     noadchinsts();
LADP     noadchinsts();
MOVP     noadchinsts();
FJPP     entjumps();
XJPP     entjumps();
ENTP     entjumps();
TNDP     oneopldst();
LOLP     oneopldst();
LDOP     oneopldst();
SRDP     oneopldst();
SROP     oneopldst();
STRP     twoopldst();
LDAP     twoopldst();
OTHERWISE: loadbyte( .op );
TESN:   read( prr );
END; ! assembler

GLOBAL ROUTINE asm( lstn ) =
BEGIN
    LOCAL maxppc;
    listing := .lstn;
    ttcrlf(.ttyno); ttoutstr(.ttyno,version);
    ttcrlf(.ttyno);
    initassembler();
    generate();
    maxppc := .ppc - 1;
    ppc := 0;
    read( prr, cn );
    generate(); cn );
    fileclose(prr);
    IF .listing THEN fileclose(lstfil);
    ttoutstr(.ttyno,mesi); ttcrlf(.ttyno);
```

05410
05420
05430
05440
05450
05460
05470
05480
05490
05500
05510
05520
05530
05540
05550
05560
05570
05580
05590
05600
05610
05620
05630
05640
05650
05660
05670
05680
05690
05700
05710
05720
05730
05740
05750
05760
05770
05780
05790
05800
05810
05820
05830
05840
05850
05860
05870
05880
05890
05900
05910
05920
05930
05940

```
059950       RETURN .maxppc
059960       END; ! .asm
059970
059980       END
059990       E?JJ0M
```

```
%***********************************************************************
%***********************************************************************
%**                                                                  **%
%**   MODULE  :: PMACHINE                                            **%
%**   DATE :: 1 JULY 1982                                            **%
%**   AUTHOR :: S SRINIVAS RAGHURAM                                  **%
%**   INPUT :: P-Machine's object code in P-Machine's               **%
%**            PCODE array.                                          **%
%**   FILES :: Files INPUT & PRD are input files                    **%
%**            and outPut & PRR are the output files                **%
%**            produced by the interpreter                          **%
%**   ENTRY :: INTR(TRCE,TTYOUT);                                    **%
%**            trce=true then execution trace is                    **%
%**            generated.                                            **%
%**            ttyout=true then the results being                   **%
%**            written on the file output are also                  **%
%**            displayed on the terminal.                           **%
%**                                                                  **%
%***********************************************************************
%***********************************************************************

MODULE pmachine ( unsafe , nooptimize ,nofinal , nolist ) =
%
Machine ERRORS:
   0     HEAP OVERFLOWS STACK
   1     ILLEGAL INSTRUCTION
   2     NOT IMPLEMENTED
   3     ERROR IN CASE STATEMENT
   4     STACK OVERFLOWS HEAP
   5     VALUE OUT OF RANGE
   6                             .
%

BEGIN
MACRO
   t1bit    =  7.1 s,          ! type1 or type2 instruction if set else type0
   t2bit    =  6.1 s,          ! type2 instruction if set
   bits6    = 00.6 s,          ! least significant six bits
   lsbv     = 00.8 s,          ! least significant byte
   msby     = 08.8 s,          ! most significant byte.
   bool     = 00.1 s,
   sign     = 15.1 s;

MACRO
   cw = pstk[ .psp - 1 ] s,    ! current word pointed by sp
   nw = pstk[ .psp + 1 ] s,    ! next word
   pw = pstk[ .psp - 3 ] s,    ! previous word
   pushi = psp = .psp + 2 s,   ! push integer
   popi  = psp = .psp - 2 s,   ! pop integer
   pops  = psp = .psp - 8 s;   ! pop set ( 4 words )

BEND
```

00010
00020
00030
00040
00050
00060
00070
00080
00090
00100
00110
00120
00130
00140
00150
00160
00170
00180
00190
00200
00210
00220
00230
00240
00250
00260
00270
00280
00290
00300
00310
00320
00330
00340
00350
00360
00370
00380
00390
00400
00410
00420
00430
00440
00450
00460
00470
00480
00490
00500
00510
00520
00530
00540

```
00550  version  =  UPLIT  ASCIZ  "PAS316 P-Interpreter Version of Jul 1,82",
00560  mes1     =  UPLIT  ASCIZ  "End of Execution",
00570  err0     =  UPLIT  ASCIZ  "HEAP OVERRUNS STACK",
00580  err1     =  UPLIT  ASCIZ  "ILLEGAL INSTRUCTION",
00590  err2     =  UPLIT  ASCIZ  "NOT IMPLEMENTED",
00600  err3     =  UPLIT  ASCIZ  "ERROR IN CASE STATEMENT",
00610  err4     =  UPLIT  ASCIZ  "STACK OVERRUNS HEAP",
00620  err5     =  UPLIT  ASCIZ  "VALUE OUT OF RANGE";
00630
00640  BIND
00650  maxcode  =  #40000             ; size of code array
00660  maxstk   =  #40000             ; size of PASCAL STACK
00670  nilvalue =  #4000
00680  expstk   =  #100               ; max size by which STACK can grow
00690                                 ; during expression evaluation
00700
00710  false    =  0
00720  true     =  1
00730  outadr   =  12
00740  ;ASCII codes
00750  zero = #60, nine = #71, nul = #0, space = #40, plus = #53, minus = #55;
00760
00770  EXTERNAL reset , rewrite;
00780  EXTERNAL get, put;
00790  EXTERNAL read , readin , readint , write , writeln;
00800  EXTERNAL endoffile , endofline;
00810  EXTERNAL allclose;
00820  EXTERNAL ttcrlf , ttputoct , ttoutstr , putlch;
00830  EXTERNAL fetchi;
00840  EXTERNAL getfilename;
00850  EXTERNAL ttyno;
00860
00870  STRUCTURE pmcstack [ i ] = [ i * BYTES ]
00880  ( .pmcstack + .i * BYTES ) <0,16>;
00890
00900  BYTE OWN pmcstack pstk [ maxstk ];
00910
00920  BYTE OWN VECTOR string [ 16 ];
00930
00940  ; P-MACHINE registers
00950
00960  OWN ppc    ;   ; Program Counter
00970      psp    ;   ; Stack Pointer ; points to top of pstk
00980      mp     ;   ; Mark Pointer ; points to beginning of a data segment
00990      np     ;   ; New Pointer ; points to top of dynamically allocated area
01000      p      ;   ; INSTRUCTION REGISTER; P = field
01010      q      ;   ; INSTRUCTION REGISTER; Q = field
01020      l      ;   ; GPR ; Low register
01030      c      ;   ; GPR ; Count register
01040      b      ;   ; GPR ; Boolean register
01050
01060
01070  BYTE OWN op ;     ; INSTRUCTION REGISTER ; OPCODE = field
01080      interpret;    ; controls execution of interpreter
```

```
OWN VECTOR cb [ 4 ] ;    ! Constant Buffer

OWN  trace, outtty;

ROUTINE pmcerr( n ) =
BEGIN
  LOCAL errmesg;
  ttcrlf(.ttyno);
  ttoutstr(.ttyno,UPLIT ASCIZ "Run Time ERROR: ");
  ttputoct(.ttyno,.n);
  ttcrlf(.ttyno);
  CASE .n OF
  SET
  errmesg = err0;
  errmesg = err1;
  errmesg = err2;
  errmesg = err3;
  errmesg = err4;
  errmesg = err5
  TES;
  ttoutstr(.ttyno,.errmesg); ttcrlf(.ttyno);
  ttoutstr(.ttyno,UPLIT ASCIZ "PPC = "); ttputoct(.ttyno,.ppc-1);
  ttoutstr(.ttyno,UPLIT ASCIZ "OP  = "); ttputoct(.ttyno,.op);
  ttoutstr(.ttyno,UPLIT ASCIZ "PSP = "); ttputoct(.ttyno,.psp-1);
  ttoutstr(.ttyno,UPLIT ASCIZ "MP  = "); ttputoct(.ttyno,.mp);
  ttoutstr(.ttyno,UPLIT ASCIZ "NP  = "); ttputoct(.ttyno,.np);
  ttcrlf(.ttyno);
  EMT(0);
END; ! pmcerr

ROUTINE locfetch =
BEGIN
  BYTE LOCAL tmp;
  tmp = fetch(.ppc); ppc = .ppc + 1;
  RETURN .tmp;
END; ! locfetch

ROUTINE qlocfetch =
BEGIN
  qvlsbv> = locfetch();
  qvmspv> = locfetch();
END; ! qlocfetch

ROUTINE base( p ) =
BEGIN
  STACKLOCAL ad;
  ad = .mp;
  WHILE .p GTR 0 DO
  BEGIN
    ad = .pstk[ .ad + 4 ];   ! static link
    p = .p - 1;
  END;
  RETURN .ad;
```

```
END; ! base

ROUTINE filadr =
( .cw - 10 ) DIV 2 ;

ROUTINE writestring( f , l , h ) =
BEGIN
IF .h GTR .l THEN
INCR i FROM 1 TO ( .h - .l ) DO
BEGIN
IF .outtty THEN IF .cw EOL outadr THEN
putich(.ttyno,space);
write( .f , space ) ;
END;
ELSE l = .n;
INCR i FROM 0 TO ( .l - l ) DO
BEGIN
IF .outtty THEN IF .cw EOL outadr THEN
putich(.ttyno,string[.i]);
write( .f , .string[ .i ] )
END;
END; ! writestring

ROUTINE coctintstr( n ) =
BEGIN
LOCAL dg;
dg = .n<sign>; string[0] = zero + .dg;
INCR i FROM 1 TO 5 DO
BEGIN
dg = .n<12,3>;
string[ .i ] = zero + .dg;
.n = .n * 8;
END;
INCR i FROM 6 TO 9 DO string[ .i ] = space;
END; ! coctintstr

ROUTINE cdecintstr( n ) =
BEGIN
BYTE LOCAL VECTOR stk[ 5 ];
BYTE LOCAL ind , sgn;
sgn = .n<sign>;
string[0]=( IF .sgn THEN minus ELSE space );
IF .sgn THEN n = -.n;
ind = -1;
DO
(ind = .ind + 1;
stk[ .ind ] = .n MOD 10;
.n = .n DIV 10;)
UNTIL .n EQL 0;
INCR i FROM .ind+2 TO 9 DO string[.i] = space;
sgn = 1; ! henceforth used as array index for string
DECR i FROM .ind TO 0 DO
BEGIN
string[ .sgn ] = zero + .stk[ .i ];
```

01630
01640
01650
01660
01670
01680
01690
01700
01710
01720
01730
01740
01750
01760
01770
01780
01790
01800
01810
01820
01830
01840
01850
01860
01870
01880
01890
01900
01910
01920
01930
01940
01950
01960
01970
01980
01990
02000
02010
02020
02030
02040
02050
02060
02070
02080
02090
02100
02110
02120
02130
02140
02150
02160

```
            sgn = .sgn + 1;
        END; ! cdecintstr

    ROUTINE compare =
        BEGIN
        pop1;
        c = .cw; l = .nw;
        c = .0; b = true;
        WHILE .b AND ( .c NEQ .q ) DO
            IF .pstk[ .p + .c ] EQL .pstk[ .l + .c ] THEN c = .c + 2
            ELSE b = false;
        END; ! compare

    ROUTINE execone =
        BEGIN ! typeq instructions : standard proc/func & no operand instructions
        MACRO relop( WHAT ) =
            BEGIN
            pop1;
            cw = .cw<bool> WHAT .nw<bool>;
            ENDS ;

        MACRO intgop( what ) =
            BEGIN
            pop1;
            cw = .cw WHAT .nw;
            ENDS ;

        MACRO b1 = 1 s , b2 = c s ;

        MACRO setop( WHAT ) =
            BEGIN
            pops;
            D1 ; psp = 7; b2 = .psp + 1;
            INCR i FROM 0 TO 3 DO
                pstk[ .b1 + 2 * .i ] = .pstk[ .b1 + 2 * .i ] WHAT .pstk[ .b2 + 2 * .i ];
            ENDS ;

    ROUTINE ex10 =          ! standard procedure/function
        BEGIN
        LOCAL opcode;
        opcode = .op;
        CASE .opcode OF
        SET
        !0 : ELN
            cw = endofline( filadr() );

        !1 : GET
            BEGIN
            get( filadr() ); pop1;
            END;
```

02170
02180
02190
02200
02210
02220
02230
02240
02250
02260
02270
02280
02290
02300
02310
02320
02330
02340
02350
02360
02370
02380
02390
02400
02410
02420
02430
02440
02450
02460
02470
02480
02490
02500
02510
02520
02530
02540
02550
02560
02570
02580
02590
02600
02610
02620
02630
02640
02650
02660
02670
02680
02690
02700

```
! 2 : RDC
BEGIN
read( filadr() , pstk[ .pw ] );
psp = .psp - 4
END;

! 3 : RDI
BEGIN
readint( filadr() , pstk[ .pw ] );
psp = .psp - 4
END;

! 4 : RDR
omcerr(2);

! 5 : RLN
BEGIN
readln( filadr() ); pop1;
END;

! 6 : PUT
BEGIN
put( filadr() ); pop1;
END;

! 7 : WRC
BEGIN
string[0] = .pstk[.psp = 5 ];
writestring( filadr() , 1 , .pw);
psp = .psp - 6;
END;

! 8 : WRI
BEGIN
cdecintstr( .pstk[.psp = 5 ] );
writestring( filadr() , 10 , .pw );
psp = .psp - 6;
END;

! 9 : WRO
BEGIN
coctintstr( .pstk[.psp = 5 ] );
writestring( filadr() , 10 , .pw );
psp = .psp - 6;
END;

! 10 : WRR
omcerr(2);

! 11 : WRS
BEGIN
writestring( filadr() , .pstk[ .psp = 5 ] , .pw );
psp = .psp - 6;
END;
```

02710
02720
02730
02740
02750
02760
02770
02780
02790
02800
02810
02820
02830
02840
02850
02860
02870
02880
02890
02900
02910
02920
02930
02940
02950
02960
02970
02980
02990
03000
03010
03020
03030
03040
03050
03060
03070
03080
03090
03100
03110
03120
03130
03140
03150
03160
03170
03180
03190
03200
03210
03220
03230
03240

```
!12 : WLN
BEGIN
  writeln( filadr() );
  IF .outtty THEN IF .cw EQL outadr THEN
    ttcrlf(.ttyno);
  pop1;
END;

!13 : NEW
BEGIN
STACKLOCAL ad;
ad = .np;  .cw;
IF .ad LEQ ( .psp +expstk ) THEN pmcerr(0);
  np = .ad;
  ad = .pw;
  pstk[ .ad ] = .np;
  psp = .psp - 4;
END;

!14 : PAK
  pmcerr(2);

!15 : RST
BEGIN
  np = .cw;
  pop1;
END;

!16 : SAV
BEGIN
  pstk[ .cw] = .np; popi;
END;

!17 : SIN
  pmcerr(2);

!18 : COS
  pmcerr(2);

!19 : ATN
  pmcerr(2);

!20 : EXP
  pmcerr(2);

!21 : LOG
  pmcerr(2);

!22 : SQT
  pmcerr(2);

!23 : UNUSED OPCODE
```

```
          ; ! nooperation

!24 : ! UNUSED OPCODE
     ; ! nooperation

!25 : ! UNUSED OPCODE
     ; ! nooperation

!26 : ! UNUSED OPCODE
     ; ! nooperation

!27 : ! UNUSED OPCODE
     ; ! nooperation

!28 : ! UNUSED OPCODE
     ; ! nooperation

!29 : ! UNUSED OPCODE
     ; ! nooperation

!30 : ! UNUSED OPCODE
     ; ! nooperation

!31 : ! UNUSED OPCODE
     ; ! nooperation
TES;
END;     ! ex10

ROUTINE ex11 =          ! boolean and integer operations
     BEGIN
     LOCAL opcode;
     opcode = .op - 32;
     CASE .opcode OF
     SET

     ! NO OPERAND INSTRUCTIONS

!32 : ! EOF
cw = endoffile( filadr() );

!33 : ! NOT
cw = NOT .cw;

!34 : ! IOR
BEGIN
     popi;
     cw= .cw OR .nw;
END;

!35 : ! AND
BEGIN
     popi;
     cw= .cw AND .nw;
END;
```

```
04330   !36: LDCBF
04340   BEGIN
04350     nw := false; pushi;
04360   END;
04370
04380   !37: LDCBT
04390   BEGIN
04400     nw := true; pushi;
04410   END;
04420
04430
04440   !38: LESB
04450   relop( LSS );
04460
04470   !39: LEQB
04480   relop( LEQ );
04490
04500   !40: EQUB
04510   relop( EQL );
04520
04530   !41: NEQB
04540   relop( NEQ );
04550
04560   !42: GEQB
04570   relop( GEQ );
04580
04590   !43: GRTR
04600   relop( GTR );
04610
04620   !44: ODD
04630   cw := .cw<bool>;
04640
04650   !45: NGI
04660   cw := -.cw;
04670
04680   !46: ADI
04690   intgop( + );
04700
04710   !47: SBI
04720   intgop( - );
04730
04740   !48: MPI
04750   intgop( * );
04760
04770   !49: DVI
04780   intgop( DIV );
04790
04800   !50: MOD
04810   intgop( MOD );
04820
04830   !51: SQI
04840   cw := .cw * .cw;
04850
04860   !52: ARI
```

```
IF .cw<sign> THEN cw = -.cw;

:53: UPCW
BEGIN
    cw = nilvalue; pushi;
END;

:54: LESA,LESI
    intgop( LSS );

:55: LEQA,LEQI
    intgop( LEQ );

:56: EQUA,EQUI
    intgop( EQL );

:57: NEQA,NEQI
    intgop( NEQ );

:58: GEQA,GEQI
    intgop( GEQ );

:59: GRTA,GRTI
    intgop( GTR );

TES;    ! ex11
END;

ROUTINE ex12 =    ! set operations
BEGIN
LOCAL opcode;
opcode = .op - 60;
CASE .opcode OF
SET

:60: SGS
BEGIN
    cw = -.cw;
    cw = nw = pstk[ .psp + 3 ] = pstk[ .psp + 5 ] = 0;
    pstk[ .psp - 1 + 2 * ( .c DIV 16 ) ] = 1 < ( .c MOD 16 );
    psp = .psp + 6;
END;

:61: INN
BEGIN
pops;
    cw = ( IF ( 1 < ( .cw MOD 16 ) AND
            .pstk[ .psp + 1 + 2 * ( .cw DIV 16 ) ] ) EOL 0 THEN false
        ELSE true );
END;

:62: UNI      A = A OR B
setop( OR );

:63: INT      A = A AND B
```

```
setop( AND );

!64: DIF                    A = A AND NOT B
setop( AND NOT );

!65: LESS
pmcerr(1);

!66: GEQS       A IS A SUBSET OF B. IF A = (A AND B)
BEGIN
psp = .psp - 14;
b1 = .psp - 1; b2 = .psp + 7;
INCR i FROM 0 TO 3 DO
BEGIN
pstk[ .b1 ] = ( .pstk[ .b1 + 2 * .i ] EOL ( .pstk[ .b1 + 2 * .i ]
              AND .pstk[ .b2 + 2 * .i ] ) );
IF NOT .pstk[ .b1 ] THEN EXITLOOP;
END
END;

!67: EQUS
BEGIN
psp = .psp - 14;
b1 = .psp - 1; b2 = .psp + 7;
INCR i FROM 0 TO 3 DO
BEGIN
pstk[ .b1 ] = ( .pstk[ .b1 + 2 * .i ] EQL .pstk[ .b2 + 2 * .i ] );
IF NOT .pstk[ .b1 ] THEN EXITLOOP;
END
END;

!68: NEQS
BEGIN
psp = .psp - 14;
b1 = .psp - 1; b2 = .psp + 7;
INCR i FROM 0 TO 3 DO
BEGIN
pstk[ .b1 ] = ( .pstk[ .b1 + 2 * .i ] NEQ .pstk[ .b2 + 2 * .i ] );
IF NOT .pstk[ .b1 ] THEN EXITLOOP;
END
END;

!69: GEQS       B IS A SUBSET OF A . IF B = ( B AND A )
BEGIN
psp = .psp - 14;
b1 = .psp - 1; b2 = .psp + 7;
INCR i FROM 0 TO 3 DO
BEGIN
pstk[ .b1 ] = ( .pstk[ .b2 + 2 * .i ] EOL ( .pstk[ .b2 + 2 * .i ] AND .pstk[ .b1 + 2 * .i ] ) );
IF NOT .pstk[ .b1 ] THEN EXITLOOP;
END
END;
```

05410
05420
05430
05440
05450
05460
05470
05480
05490
05500
05510
05520
05530
05540
05550
05560
05570
05580
05590
05600
05610
05620
05630
05640
05650
05660
05670
05680
05690
05700
05710
05720
05730
05740
05750
05760
05770
05780
05790
05800
05810
05820
05830
05840
05850
05860
05870
05880
05890
05900
05910
05920
05930
05940

```
05950        !70 : GRTS
05960           omcerr(1);
05970         IFS:    !ex12
05980         END;    !ex12
05990
06000   ROUTINE ex13 =   ! real and rest of the operations
06010     BEGIN
06020     LOCAL opcode;
06030     opcode = .op - 71;
06040     CASE .opcode OF
06050     SET
06060        !71: TRC
06070        omcerr(2);
06080
06090        !72: FUO
06100        omcerr(2);
06110
06120        !73: FLT
06130        omcerr(2);
06140
06150        !74: NGR
06160        omcerr(2);
06170
06180        !75: ADR
06190        omcerr(2);
06200
06210        !76: SBR
06220        omcerr(2);
06230
06240        !77: MPR
06250        omcerr(2);
06260
06270        !78: OVR
06280        omcerr(2);
06290
06300        !79: SQR
06310        omcerr(2);
06320
06330        !80: ABR
06340        omcerr(2);
06350
06360        !81: LESR
06370        omcerr(2);
06380
06390        !82: LEQR
06400        omcerr(2);
06410
06420        !83: EQUR
06430        omcerr(2);
06440
06450        !84: NEQR
06460        omcerr(2);
06470
06480
```

```
!85: GEQR
  omcerr(2);

!86: GRTR
  omcerr(2);

!87 : TJC
  omcerr(3);

!88: STP
  interpret = false;
IES;
IF .OP LEO 31 THEN ex13
ELSE IF .OP LEO 59 THEN ex10()
ELSE IF .OP LEO 70 THEN ex11()
ELSE ex13();          ex12()
END; !execone

ROUTINE exectwo =
BEGIN
ROUTINE ex20 =   ! load store and block rel ops.
BEGIN
LOCAL opcode;
opcode = .op - 128;
CASE .opcode OF
SET !128: LDC
BEGIN
INCR i FROM 0 TO .c DO
  pstk[ .psp + 2 * ( .c + 1 )] = .cbl .i ];
  psp = .psp + 2 * ( .c + 1 );
END.

!129: IND
BEGIN
q = .cw + .q;
INCR i FROM 0 TO .c DO
  pstk[ .psp - 1 + 2 * .i ] = .pstk[ .q + 2 * .i ];
  psp = .psp + 2 * .c;
END;

!130: LDL,LDO,LOD
BEGIN
INCR i FROM 0 TO .c DO
  pstk[ .psp + 2 * ( .c + 1 )] = .pstk[ .q + 2 * .i ];
  psp = .psp + 2 * ( .c + 1 );
END;

!131: STO
BEGIN
LOCAL locpsp;
locpsp = .psp - 2 * ( .c + 1 ) - 2;
psp = .locpsp;
```

```
06490
06500
06510
06520
06530
06540
06550
06560
06570
06580
06590
06600
06610
06620
06630
06640
06650
06660
06670
06680
06690
06700
06710
06720
06730
06740
06750
06760
06770
06780
06790
06800
06810
06820
06830
06840
06850
06860
06870
06880
06890
06900
06910
06920
06930
06940
06950
06960
06970
06980
06990
07000
07010
07020
```

51

```
q = .qw;
INCR i FROM 0 TO .c DO
  .pstk[ .osp + 2 * .i ] = .pstk[ .osp + 3 + 2 * .i ];
END;
!132: STL,SRO,STR
BEGIN
LOCAL locpsp;
locpsp = .psp - 2 * ( .c + 1 );
psp = .locpsp;
INCR i FROM 0 TO .c DO
  .pstk[ .q + 2 * .i ] = .pstk[ .psp + 1 + 2 * .i ];
END;
!133: LESM
BEGIN
compare();
cw = ( .pstk[ .p + .c ] LSS .pstk[ .l + .c ] ) AND NOT .b;
END;
!134: LEQM
BEGIN
compare();
cw = ( .pstk[ .p + .c ] LEQ .pstk[ .l + .c ] ) OR .b;
END;
!135: EQUM
BEGIN
compare();
cw = .b;
END;
!136: NEQM
BEGIN
compare();
cw = NOT .b;
END;
!137: GEQM
BEGIN
compare();
cw = ( .pstk[ .p + .c ] GEQ .pstk[ .l + .c ] ) OR .b;
END;
!138: GRTM
BEGIN
compare();
cw = ( .pstk[ .p + .c ] GTR .pstk[ .l + .c ] ) AND NOT .b;
END;
TES;
END;    ! ex20

ROUTINE ex21 =
```

```
        psp := .psp - 4;
        INCR i FROM 0 TO .q - 1 BY 2 DO
            pstk[ .p + .i ] := .pstk[ .i + .i ];
        END;

!149: LDCI
BEGIN
    nw := .q; push1;
END;

!150: LDA
BEGIN
    nw := base( .p ) + .q;
    push1;
END;

!151: CUP
BEGIN
    mp := .psp := ( .p + 9 );
    pstk[ .mp + 8 ] := .ppc;
    ppc := .q;
END;

!152: MST
BEGIN
    pstk[ .psp + 5 ] := base( .p );
    pstk[ .psp + 7 ] := .mp;
    psp := .psp + 10;
END;

!153: RET
BEGIN
    psp := ( IF .p EQL 0 THEN .mp - 1
            ELSE IF .p EQL 2 THEN .mp + 3
                ELSE .mp + 1 ) );
    ppc := .pstk[ .mp + 8 ];
    mp := .pstk[ .mp + 6 ];
END; ! executwo

!154: CHK
IF ( .cw LSS .1 ) OR ( .cw GTR .q ) THEN pmcerr(5);

!155: LCA
; !NO OPERATION

IES;
END; ! ex21
IF .op LEQ 138 THEN ex20()
ELSE ex21();
END; ! exectwo

ROUTINE type1 =
BEGIN
    IF .op LEQ 149 THEN    ! group 3
```

```
! LFSM,LEOM,EQOM,NEQM,GEQM,GRTM,DEC,INC,LDCI
!.IXA,LAD,LAL,FJP,XJP,UJP,ENT,MOV,LDCI
   qlocfetch() LEO 151 THEN  ! group 4
ELSE IF op.OP BEGIN ! LDA,CUP
   p := 0;
      p<lsby> = locfetch();
      qlocfetch();
   END
ELSE IF op LEO 153 THEN  ! group 2
   BEGIN ! MST,RET
   p := 0;
      p<lsby> = locfetch();
   END
ELSE IF op EQL 154 THEN  ! group 5
   BEGIN ! CHK
   1<lsby> = locfetch();
   1<msby> = locfetch();
   qlocfetch();
   END
ELSE IF op EQL 155 THEN  ! group 6
   BEGIN ! LCA
   c := 0; p := 0;
   p<lsby> = locfetch();
   WHILE .p<lsby> NEQ nul DO
   BEGIN
   string[ .c ] = .p<lsby>;
   c := .c + 1;
   p<lsby> = locfetch();
   END;
   END;
   exectwo();
END; ! type1

ROUTINE type2execzero =
BEGIN
MACRO
m0( count , nop ) =
   BEGIN
   c := count; op = nop;
   END $ ,
m1( count , nop ) =
   BEGIN
   c := count; qlocfetch(); op = nop;
   END $ ,
m2( count , nop ) =
   BEGIN
   c := count; qlocfetch(); a := .mp + .a; op = nop;
   END $ ,
m3( count , nop ) =
   BEGIN
```

```
c := count; p := 0; p<lsby> := locfetch(); qlocfetch();
q := base( .b ) + .a;
op := nop;
END s ,

m4( count , nop ) =
BEGIN
INCR i FROM 0 TO count DO
BEGIN
cbl .i ]<lsby> := locfetch();
cbl .i ]<msby> := locfetch();
END;
c := count; op := nop;
END s;

CASE .op<bits6> OF
SET
[192]: LDCK        group 6
[193]: LDCS
[4(3]: -128 };
[104]: INDA,INDI,INDb    group 3
[105]: -129 };
[106]: INDR
[107]: INDS
[107]: LDLA,LDLI,LDLB    group 3
[208]: -130 };
[209]: LDLR
[2(1]: LDLS
[2(3]: -130 };

[200]: LDDA,LDDI,LDDB    group 3
[201]: LDDR    -130 };
[202]: LDDS
[203]: -130 };

[203]: LDOA,LODI,LODB    group 4
[204]: LODR    -136 };
[205]: LODS
[206]: -130 };

[206]: STDA,STDI,STDB    group 1
[207]: STOR    -131 };
[208]: STDS
[203]: -131 };
```

```
:209: STLA,STLT,STLB    group 3
.220(0 ,132 ) ;
.2210: STLR
.2211: STLS  ,132 ) ;
.220(3 ,132 ) ;

:212: SRDA,SRDI,SRDB    group 3
.210(0 ,132 ) ;
.2213: SRDR
.2214: SRDS  ,132 ) ;
.210(3 ,132 ) ;

:215: STRA,STRI,STRB    group 4
.230(0 ,132 ) ;
.2216: STRR
.230(1 ,132 ) ;
.2217: STRS
.230(3 ,132 ) ;
.IES:
exectwo() ;
END; ! type2execzero

ROUTINE initinterpreter =
BEGIN
LOCAL trk,srfc,sectr;
BIND
    input = 0 , inpadr = 10 ,
    output = 1 , outadr = 12 ,
    prd = 2 , prdadr = 14 ,
    prr = 3 , prradr = 16 ,

ppc = 0 ; psp = -1 ; mp = 0 ; np = maxstk; ! checkup np initial value
interpret = true;
ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "input");
getfilename(trk,srfc,sectr); ttoutstr(.ttyno,.trk,.srfc,.sectr);
reset(input,pstk[inpadr],.trk,.srfc,.sectr);
ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "prd");
getfilename(trk,srfc,sectr);
reset(prd ,pstk[prdadr],.trk,.srfc,.sectr);
getfilename(trk,srfc,sectr); ttoutstr(.ttyno,UPLIT ASCIZ "Output");
rewrite(output,pstk[outadr],.trk,.srfc,.sectr);
ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "Prr");
getfilename(trk,srfc,sectr);
rewrite(prr ,pstk[prradr],.trk,.srfc,.sectr);
IF .trace THEN   PSP STK[sp]     PPC   Opcode"); ttcrlf(.ttyno);
BEGIN
ttoutstr(.ttyno,UPLIT ASCIZ "
END; ! initinterpreter
```

```
10270   GLOBAL ROUTINE intr(trce,ttyout) =
10280     BEGIN
10290     ttcrlf(.ttyno);
10300     ttoutstr(.ttyno,version); ttcrlf(.ttyno);
10310     trace = .trce; outtty = .ttyout;
10320     initinterpreter();
10330     WHILEW .interpret DO
10340       BEGIN
10350       op = locfetch();
10360       IF .trace THEN
10370         BEGIN
10380         ttputoct(.ttyno,.psp-1); putlch(.ttyno,space); ttputoct(.ttyno,.pstkl.psp-1); putlch(.tt
10390         ttputoct(.ttyno,.pstkl.psp-5]); putlch(.ttyno,space); ttputoct(.ttyno,.pw); putlch(.ttyno
10400         ttputoct(.ttyno,.cw); putlch(.ttyno,space);
10410         ttputoct(.ttyno,.ppc); putlch(.ttyno,space); ttputoct(.ttyno,.op); ttcrlf(.ttyno); NOP();
10420         END;
10430       IF NOT .op<t1bit> THEN execone() ! group 1
10440       ELSE IF .op<t2bit> THEN type2execzero()
10450       ELSE type1()
10460       END;
10470     allclose();
10480     ttoutstr(.ttyno,mes1)
10490     END; ! intr
10500
10510   END
10520   ELUDOM
```

```
%****************************************************************************
 ****************************************************************************
 ****                                                                    ****
 ****        MODULE : SYSTEMLIBRARY                                      ****
 ****        DATE   : 1 JULY 1982                                        ****
 ****        AUTHOR : S SRINIVAS RAGHURAM                                ****
 ****        ENTRIES: TTCRLF(TTYLIN); TTPUTOCT(TTYLIN,VAL);             ****
 ****                 TTGETDEC(TTYLIN,VAL); INITTAPS();                  ****
 ****                 TTOUTSTR(TTYLIN,ADDR); SYSERR(N);                  ****
 ****                 GETFILENAME(TRK,SRFC,SECTR);                       ****
 ****                 TYPE(TRK,SRFC,SECTR);                              ****
 ****                 COPY(ST,SC,SS,DT,DC,DS);                           ****
 ****                 PSAVE(TRK,SRFC,SECTR,MPC);                         ****
 ****                 PLOAD(TRK,SRFC,SECTR);                             ****
 ****                                                                    ****
 ****************************************************************************
 MODULE syslib(nolist)=
 BEGIN
 EXTERNAL getlch,putlch;
 EXTERNAL ttvno;
 EXTERNAL read,write,readln,writeln;
 EXTERNAL reset,rewrite,endofline,endoffile,fileclose;
 EXTERNAL open,close,getbyte,putbyte;
 EXTERNAL fetch,deposit;

 BIND space = #40, nul = #0;
 BIND rd = 0, wr = 1, input = 0, output = 1;
 BIND false = 0, true = 1;

 MACRO lsby = 0,8$, msby = 8,8$;

 GLOBAL ROUTINE ttcrlf(ttylin)=
 BEGIN putlch(.ttylin,#12);putlch(.ttylin,#15);
 END;

 GLOBAL ROUTINE ttputoct(ttylin,val)=
 BEGIN
 LOCAL ch;
 ROUTINE octch(ttylin,ch)=
     putlch(.ttylin,#60+.ch);
 ch=.val<15,1>;octch(.ttylin,.ch);
 val=.val*2;
 INCR i FROM 1 TO 5 DO
     (ch=.val<13,3>;octch(.ttylin,.ch);
 val=.val*8);
 END;

 GLOBAL ROUTINE ttgetdec(ttylin,val) =
     ! val is a refrence parameter
 BEGIN
 LOCAL ch , i , n;
 n = 0; i = 0;
```

```
            getlch(.ttylin,ch);
    WHILE(.ch GEQ #71 AND .ch GEQ #60 AND .i LEQ 5 )DO
    BEGIN
        n = .n * 10 + .ch - #60;
        i = .i + 1;
        getlch(.ttylin,ch) ;
    END;
    IF .ch EQL #12 THEN putlch(.ttylin,#15)
    ELSE IF .ch EQL #15 THEN putlch(.ttylin,#12);
    val = .n;
    END; .! ttgetdec

GLOBAL ROUTINE ttoutstr(ttylin,addr) =
                ! addr is a reference parameter
    BEGIN
    WHILE .(.addr)<0,8> NEQ nul DO
        BEGIN
        putlch(.ttylin,.(.addr)<0,8>); addr = .addr + 1 ;
        END
    END;

ROUTINE EMT emttrap =
    BEGIN
    BIND progcnt = #177740, stkptr = #177742;
    EXTERNAL getcommand,savstkval;
    stkptr = .savstkval;
    progcnt = .getcommand;
    END; .! emttrap

ROUTINE INTERRUPT oddadrtrap =
    BEGIN
    ttoutstr( .ttyno, UPLIT ASCIZ "ODD ADDRESS / ILL. INSTR. TRAP AT PC: ");
    ttoutoct( .ttyno, .OLDPC );
    EMT(0)
    END; .! oddadrtrap

ROUTINE INTERRUPT stkovrfltrap =
    BEGIN
    ttoutstr( .ttyno, UPLIT ASCIZ "SYSTEM STACK OVERFLOW AT PC: ");
    ttoutoct( .ttyno, .OLDPC );
    EMT(0)
    END; .! stkovrfltrap

ROUTINE INTERRUPT sometrap =
    BEGIN
    ttoutstr( .ttyno, UPLIT ASCIZ "SOME TRAP AT PC: ");
    ttoutoct( .ttyno, .OLDPC );
    EMT(0)
    END; .! sometrap

GLOBAL ROUTINE inittraps =
    BEGIN
    #20 = emttrap; #34 = oddadrtrap; #4 = stkovrfltrap;
    #22 = 0; #36 = 0; #6 = 0;
```

```
    #10 = sometrap; #14 = sometrap; #24 = sometrap;
    #12 = 0;        #16 = 0;        #26 = 0;
    #30 = sometrap; #16 = sometrap; #40 = sometrap;
    #32 = 0;        #16 = 0;        #42 = 0;
    #00 = sometrap; #02 = 0;
    END; := inittraps

GLOBAL ROUTINE getfilename(trk,srfc,sectr)=
    ! trk,srfc,sectr are reference parameters

BEGIN
BIND
    mes1 = UPLIT ASCIZ "File name: ",
    mes2 = UPLIT ASCIZ "Track  No.: ",
    mes3 = UPLIT ASCIZ "Surface No.: ",
    mes4 = UPLIT ASCIZ "Sector No.: ",
    mes5 = UPLIT ASCIZ "ERROR! Retype",

ROUTINE pringnum(llim , hlim)=
BEGIN
    LOCAL n,flag;
    flag = true;
    WHILE .flag DO
    BEGIN
        n = 0;
        ttgetdec(.ttyno,n);
        !$debug% putlch(.ttyno,space); ttputoct(.ttyno,.n);
        IF .n GEO .llim AND .n LEO .nlim THEN flag = false
        ELSE ttoutstr(.ttyno,mes5);
        ttcrlf(.ttyno);
    END;
    RETURN .n;
END; ! pringnum

ttcrlf(.ttyno); ttoutstr(.ttyno,mes1); ttcrlf(.ttyno);
ttoutstr(.ttyno,mes2); .trk=pringnum(100,200);
ttoutstr(.ttyno,mes3); .srfc=pringnum(0,9);
ttoutstr(.ttyno,mes4); .sectr=pringnum(1,10);
END; getfilename

GLOBAL ROUTINE type(trk,srfc,sectr)=
BEGIN
    LOCAL padr,ch,count;
    count = 0;
    reset(input,padr(.trk,.srfc,.sectr));
    UNTIL endoffile(input) DO
    BEGIN
        IF endofline(input) THEN
        BEGIN
            readln(input); ttcrlf(.ttyno); count = .count + 2
        END
        ELSE
        BEGIN
            read(input,ch); putlch(.ttyno,.ch); count = .count + 1
        END;
    END;
```

```
01090
01100
01110
01120
01130
01140
01150
01160
01170
01180
01190
01200
01210
01220
01230
01240
01250
01260
01270
01280
01290
01300
01310
01320
01330
01340
01350
01360
01370
01380
01390
01400
01410
01420
01430
01440
01450
01460
01470
01480
01490
01500
01510
01520
01530
01540
01550
01560
01570
01580
01590
01600
01610
01620
```

```
        END;
        fileclose(input);
        ttcrlf(.ttyno);
        ttoutstr(.ttyno,UPLIT ASCIZ "File size: ");
        ttputoct(.ttyno,.count + 1); putlch(.ttyno,"B");
        ttcrlf(.ttyno)
END; ! type

GLOBAL ROUTINE copy(st,sc,ss,dt,dc,ds)=
        ! copy of ASCII files only!
BEGIN
        LOCAL inpbadr,outbadr,ch;
        reset(input,inpbadr,.st,.sc,.ss);
        rewrite(output,outbadr,.dt,.dc,.ds);
        UNTIL endoffile(input) DO
        BEGIN
        IF endofline(input) THEN
                BEGIN
                readln(input); writeln(output)
                END
        ELSE
                BEGIN
                read(input,ch); write(output,.ch)
                END
        END;
        fileclose(input); fileclose(output)
END; ! copy

GLOBAL ROUTINE syserr( n )=
BEGIN
        IF .n LEQ 11 THEN
        ttoutstr(.ttyno,UPLIT ASCIZ "DISK ERROR: ")
        ELSE
        ttoutstr(.ttyno,UPLIT ASCIZ "Pascal File IO ERROR: ")
        ttputoct(.ttyno,.n);
        EMT(0);
END; ! syserr

        DISK ERROR    1 :  write attempted on a write protected disk
        DISK ERROR    2 :  seek/search error , non data transfer error
        DISK ERROR    3 :  bus grant late
        DISK ERROR    4 :  crcc error/read or write disable
        DISK ERROR    5 :  synchronous error
        DISK ERROR    6 :  drive unsafe
        DISK ERROR    7 :  command check
        DISK ERROR   10 :  sector over run
        DISK ERROR   11 :  track over run
        DISK ERROR   12 :  non existant memory
        DISK ERROR   13 :  other errors
        PASCAL IO ERROR14 :  attempt to get/read beyond EOF
        PASCAL IO ERROR15 :  buffers exhausted cannot open file
        PASCAL IO ERROR16 :  file has not been opened for write
        PASCAL IO ERROR17 :  input data erroneous
```

016300
016400
016500
016700
016800
016900
017000
017100
017200
017300
017400
017500
017600
017700
017800
017900
018000
018100
018200
018300
018400
018500
018600
018700
018800
018900
019000
019100
019200
019300
019400
019500
019600
019700
019800
019900
020000
020100
020200
020300
020400
020500
020600
020700
020800
020900
021000
021100
021200
021300
021400
021500
021600

```
PASCAL 10 ERROR20 : too many digits for an integer

GLOBAL ROUTINE psave(trk,srfc,sectr,mpc) =
BEGIN
LOCAL fsp,flag,ch;
open(.trk,.srfc,.sectr,wr,fsp,flag);
IF .flag THEN syserr(13)
ELSE
BEGIN
ttoutstr(.ttyno,UPLIT ASCIZ "Saving");
ch = .mpc<lsbv>; putbyte(.fsp,.ch);
ch = .mpc<msbv>; putbyte(.fsp,.ch);
INCR ppc FROM 0 TO .mpc DO
putbyte(.fsp,.fetch(.ppc));
close(.fsp); ttcrlf(.ttyno);
END;
END; ! psave

GLOBAL ROUTINE pload(trk,srfc,sectr) =
BEGIN
LOCAL fsp,flag,ch,mpc;
open(.trk,.srfc,.sectr,rd,fsp,flag);
IF .flag THEN syserr(13)
ELSE
BEGIN
ttoutstr(.ttyno,UPLIT ASCIZ "Loading");
getbyte(.fsp,ch); mpc<lsbv> = .ch;
getbyte(.fsp,ch); mpc<msbv> = .ch;
INCR ppc FROM 0 TO .mpc DO
BEGIN
getbyte(.fsp,cn); deposit(.ppc,.ch)
END;
close(.fsp); ttcrlf(.ttyno)
END;
END; ! pload

END
ELUDOM
```

02170
02180
02190
02200
02210
02220
02230
02240
02250
02260
02270
02280
02290
02300
02310
02320
02330
02340
02350
02360
02370
02380
02390
02400
02410
02420
02430
02440
02450
02460
02470
02480
02490
02500
02510
02520
02530
02540
02550
02560

```
%*************************************************
 *************************************************
 *************************************************
 *************************************************
 *************************************************
 *     MODULE  : PASCALFILEOPERATORS            **
 *     DATE :  1 JULY 1982                       **
 *     AUTHOR : S. SRINIVAS RAGHURAM             **
 *     ENTRIES : GET(F); PUT(F); READ(F,CH); READLN(F); **
 *               RESET(F,BADR,TRK,SRC,SECTR);    **
 *               REWRITE(F,BADR,TRK,SRC,SECTR);  **
 *               WRITE(F,CH); WRITELN(F); READINT(F,V); **
 *               ENDOFFILE(F); ENDOFLINE(F);     **
 *               FILECLOSE(F);                   **
 *************************************************%
 *************************************************

MODULE pascalfileoperators (nolist) =
BEGIN
; fields in the filetable
                        ! file pointer
fsp    = 0 ,
eoln   = 1 ,
eof    = 2 ,            ; used for taking care of look ahead char for EOLN
auxch  = 3 ,            ; address of the one char file buffer on the stack
bufadr = 4 ,
; file names code
input = 0, output = 1 , prd = 2 , prr = 3 ,
; constants
lf = #12 , cr = #15 , sp = #40 , eofch = #32 , false = 0 , true = 1 ;

EXTERNAL getbyte , putbyte , open , close , syserr ;

STRUCTURE filetable [fileno , field ] =
[ fileno*field* bytes ]
CASE .field OF
SET .field OF
(( .filetable + .fileno * bytes * field     )<0,8*bytes>; ! fsp
(( .filetable + .fileno * bytes * field + 1 )<0,8*bytes>; ! eoln
(( .filetable + .fileno * bytes * field + 2 )<0,8*bytes>; ! eof
(( .filetable + .fileno * bytes * field + 3 )<0,8*bytes>; ! auxch
(( .filetable + .fileno * bytes * field + 4 )<0,16>;      ! bufadr
TES;

BYTE OWN filetable ftl 6 , 8 1;! 8 MUST BE 6 ,BUT PUT TO AVOID JSR PC,MUL

GLOBAL ROUTINE get( f ) =
BEGIN
IF .ftl[ .f , eof ] THEN syserr(12)
; attempt to get/read beyond EOF
ELSE
BEGIN
IF .ftl[ .f , auxch ] EQL lf THEN
getbyte( .ftl[ .f , .ftl[ .f , bufadr ] )
;RISK : badr passed to reset must be a WORD address else ...
ELSE
BEGIN
```

00010
00020
00030
00040
00050
00060
00070
00080
00090
00100
00110
00120
00130
00140
00150
00160
00170
00180
00190
00200
00210
00220
00230
00240
00250
00260
00270
00280
00290
00300
00310
00320
00330
00340
00350
00360
00370
00380
00390
00400
00410
00420
00430
00440
00450
00460
00470
00480
00490
00500
00510
00520
00530
00540

```
            .ft[ .f , bufadr ] = .ft[ .f , auxch ];
            .ft[ .f , auxch ] = lf;
        END;
        IF .ft[ .f , bufadr ] EOL eofch THEN
            ft[ .f , eof ] = true; ft[ .f , eoln ] = true;
            RETURN;
        END
        ELSE
        IF .ft[ .f , bufadr ] EOL cr THEN
            BEGIN
            LOCAL ch;
            getbyte(.ft[.f , fsp ] , ch);
            ft[ .f , auxch ] = .ch;
            IF .ch EOL lf THEN
                BEGIN
                .ft[ .f , bufadr ] = sp;
                .ft[ .f , eoln ] = true;
                RETURN;
                END
            END;
        IF .ft[ .f , eoln ] THEN ft[ .f , eoln ] = false;
END; ! get

GLOBAL ROUTINE reset( f , badr , trk , srfc , sectr ) =
    BEGIN
    LOCAL tmp , flag;
    BIND rd = 0 , flag;
    ft[ .f , auxch ] = lf;
    ft[ .f , eoln ] = false;
    ft[ .f , eof ] = false;
    ft[ .f , bufadr ] = .badr;
    open( .trk , srfc , sectr , rd , tmp , flag );
    ft[ .f , fsp ] = .tmp;
    IF NOT .flag THEN get(.f) ELSE syserr(13);
    ! buffers exhausted cannot open file
END; ! reset

GLOBAL ROUTINE rewrite( f , badr , trk , srfc , sectr ) =
    BEGIN
    LOCAL tmp , flag;
    BIND wrt = 1 , flag;
    ft[ .f , auxch ] = lf;
    ft[ .f , eoln ] = true;
    ft[ .f , eof ] = true;
    ft[ .f , bufadr ] = .badr;
    open( .trk , srfc , sectr , wrt , tmp , flag );
    ft[ .f , fsp ] = .tmp;
    IF .flag THEN syserr(13);
    ! buffers exhausted cannot open file
END; ! rewrite

GLOBAL ROUTINE put(f)=
```

```
BEGIN
IF .ft[ .f , eof ] THEN
  .putbyte(.ft[.f , fsp ] , ..ft[ .f , bufadr ] )
ELSE syserr(14);    ! file has not been opened for write
END; ! put

GLOBAL ROUTINE readln(f)=
BEGIN
WHILE NOT .ft[ .f , eoln ] DO get(.f);
IF NOT .ft[.f , eof ] THEN get(.f);
END; ! readln

GLOBAL ROUTINE writeln(f)=
BEGIN
.ft[ .f , bufadr ] = cr ; put(.f);
.ft[ .f , bufadr ] = lf ; put(.f);
END; ! writeln

GLOBAL ROUTINE read(f,ch)=
     ! ch is a reference parameter
BEGIN
IF NOT .ft[ .f , eof ] THEN
  BEGIN .ft[.f , bufadr ];
    .ch = ..ft[.f]; buf get(.f);
  END
ELSE syserr(12)
     ! attempt to get/read beyond EOF
END; ! read

GLOBAL ROUTINE write(f,ch)=
BEGIN
.ft[ .f , bufadr ] = .ch;
.put(.f);
END; ! write

GLOBAL ROUTINE readint( f , v ) =
     ! v is a reference parameter
BEGIN
LOCAL
  k ,        ! digit count
  n ,        ! current value of integer
  s ;        ! sign information
BIND space = #40, plus = #53, minus = #55, zero = #60, nine = #71;
BIND nmax = 5; ! maximum no. of digits in an integer
BIND buf = .ft[.f , bufadr ];

k = 0; n = 0; s = 0;
WHILE .buf EQL space DO get(.f);
IF .buf EQL plus THEN get(.f)
ELSE IF .buf EQL minus THEN
  BEGIN
    get( .f ); s = 1;
```

```
01630        END;
01640        WHILE .buf GEQ zero AND .buf LEQ nine DO
01650        BEGIN
01660          n = 8 * .n + 2 * .n + .buf - zero;
01670          k = .k + 1;
01680          get( .f );
01690        END;
01700        IF .k EQL 0 THEN syserr(15); ! input data erroneous
01710        IF .k GTR nmax THEN syserr(16); !;too many digits for an integer
01720        !;NOTE: for integers between +/-32768 to +/-99999 results are all balls*
01730        IF .s THEN n = -.n;
01740        .v = .n;
01750      END; ! readint
01760
01770  GLOBAL ROUTINE endoffile(f) = .ft[ .f , eof ];
01780
01790  GLOBAL ROUTINE endofline(f) = .ft[ .f , eoln];
01800
01810  GLOBAL ROUTINE fileclose(f) = close( .ft[ .f , fsp ]);
01820
01830  %debug
01840  GLOBAL ROUTINE dmpfiltab(f)=
01850  BEGIN
01860    EXTERNAL ttputoct,ttcrlf,putlch;
01870    EXTERNAL ttyno;
01880    MACRO ol(what)=ttputoct(.ttyno,.ft[.f,what]) $,
01890      ol(fsp);ol(eoln);ol(eof);p1;ol(auxch);p1;
01900      ol(bufadr);ttcrlf(.ttyno);
01910    END; ! dmpfiltab
01920  debug%
01930
01940
01950
01960  end
01970  eludom
```

```
%***********************************************************************
 ***********************************************************************
 ***********************************************************************
 ****                                                               ****
 ****    MODULE :: PSEUDOFILER                                      ****
 ****    DATE   :: 1 JULY 1982                                      ****
 ****    AUTHOR :: S SRINIVAS RAGHURAM                              ****
 ****    ENTRIES:: GETBYTE(FSP,CH); PUTBYTE(FSP,CH);               ****
 ****             OPEN(TRACK,SURFACE,SECTOR,FOR,FSP,FLAG);         ****
 ****             CLOSE(FSP); INITBUFS(); ALLCLOSE();              ****
 ***********************************************************************
 ***********************************************************************
 ***********************************************************************%

MODULE pseudofiler( nolist )=
BEGIN
MACRO
purpose = 8,8 s ,
bufptr  = 0,8 s ,
trk     = 8,8 s ,
srfc    = 4,4 s ,
str     = 0,4 s ;

BIND
! fields of openfilestable
baseadr  = 0 ,
curadr   = 1 ,
purbuf   = 2 ,
nofs     = 6 ,        ! number of open files (max)
freeslot = 00 ,
rd       = 0 ,
wr       = 1 ,
undef    = 2 ,
false    = 0 ,
true     = 1 ,
eofch    = #326 ,
bufsize  = #206 ;  ! buffer size = sector size ;(words)

STRUCTURE openfilestab[ fp , j ] =
CASE .j OF
SET
  ( .openfilestab + .fp * 8     );   ! baseadr
  ( .openfilestab + .fp * 8 + 2 );   ! curadr
  ( .openfilestab + .fp * 8 + 4 );   ! purbuf
TES;

STRUCTURE buftab[ bufno , index ] =
[ .bufno * index * bytes ,] index j =
( .buftab + .bufno * 2 * bufsize * bytes + .index * bytes )<0,8*bytes>;

OWN openfilestab opnf[ nofs * 4 ];

BYTE OWN buftab bfr[ nofs , 2 * bufsize ];   ! 4 ; ! 4 ==> 3 , TO GET OVER MUL IN CODE

EXTERNAL getblk , putblk ;

%debug
```

```
00010
00020
00030
00040
00050
00060
00070
00080
00090
00100
00110
00120
00130
00140
00150
00160
00170
00180
00190
00200
00210
00220
00230
00240
00250
00260
00270
00280
00290
00300
00310
00320
00330
00340
00350
00360
00370
00380
00390
00400
00410
00420
00430
00440
00450
00460
00470
00480
00490
00500
00510
00520
00530
00540
```

```
EXTERNAL ttcrlf,ttputoct;
EXTERNAL ttvno;
debug%

ROUTINE inccuradr( fsp ) =
BEGIN
BIND k = opnf[ .fsp, curadr ];
K<str> = .K<str> + 1;
IF .K<str> GTR 10 THEN
  BEGIN
  K<str> = 1;
  K<srfc> = .K<srfc> + 1;
  IF .K<srfc> GTR 9 THEN
    BEGIN
    K<srfc> = 0;
    K<trk> = .K<trk> + 1
    END
  END
END; ! inccuradr

GLOBAL ROUTINE getbyte( fsp, ch ) =
! ch is a reference parameter
BEGIN
BIND k = opnf[ .fsp, purbuf ];
IF .K<purpose> EQL rd THEN
  BEGIN
  K<bufptr> = .K<bufptr> + 1;
  IF .K<bufptr> EQL 0 THEN
    BEGIN
    BIND j = opnf[ .fsp, curadr ];
    getblk( .j<trk> , .j<srfc> , .j<str> , bufsize , bfrl[ .fsp , 0 ] );
    inccuradr( .fsp )
    END;
  .ch = .bfrl[.fsp, .K<bufptr> ]
  END
END; ! getbyte

GLOBAL ROUTINE putbyte( fsp , ch ) =
BEGIN
BIND k = opnf[ .fsp, purbuf ];
IF .K<purpose> EQL wr THEN
  BEGIN
  bfrl[ .fsp, .K<bufptr> ] = .ch;
  K<bufptr> = .K<bufptr> + 1;
  IF .K<bufptr> EQL 0 THEN
    BEGIN
    BIND j = opnf[ .fsp, curadr ];
    putblk( .j<trk> , .j<srfc> , .j<str> , bufsize , bfrl[ .fsp , 0 ] );
    inccuradr( .fsp )
    END
  END
END; ! outbyte
```

```
00550
00560
00570
00580
00590
00600
00610
00620
00630
00640
00650
00660
00670
00680
00690
00700
00710
00720
00730
00740
00750
00760
00770
00780
00790
00800
00810
00820
00830
00840
00850
00860
00870
00880
00890
00900
00910
00920
00930
00940
00950
00960
00970
00980
00990
01000
01010
01020
01030
01040
01050
01070
01080
```

69

```
GLOBAL ROUTINE open( track, surface, sector, for, fsp, flag ) =
! fsp and flag are reference parameters ; flag = true ==> buffers
! are not available
BEGIN
LOCAL ptr , adr;
adr<trk> = .track;    adr<srfc> = .surface;
adr<str> = .sector;   ptr = nofs;
INCR K FROM 0 TO nofs-1 DO
   BEGIN
   BIND T = .opnfl[.K , baseadr ];
   IF T EQL .adr THEN
      BEGIN
      ptr = .K; EXITLOOP
      END;
   IF T EQL freeslot THEN ptr = .K
   END;
IF .ptr EQL nofs THEN
   BEGIN
   flag = true; RETURN
   END
ELSE
   BEGIN
   flag = false; .fsp = .ptr;
   opnfl[ .ptr , baseadr ] = .adr;
   opnfl[ .ptr , curadr ] = .adr;
   opnfl[ .ptr , purbuf ] = 
   opnfl[ .ptr , <purpose> ] = .for;
   opnfl[ .ptr , <buffer> ] = (IF .for EQL rd THEN 255 ELSE 0);
   END
END; ! open

GLOBAL ROUTINE close(fsp) =
BEGIN
IF .opnfl[ .fsp , purbuf ]<purpose> EQL wr THEN
   BEGIN
   BIND K = opnfl[ .fsp , curadr ];
   putbyte( .fsp, eofch );
   putblk( .K<trk> , .K<srfc> , .K<str> , bufsize , btrl , .fsp , 0 )
   END;
opnfl[ .fsp , baseadr ] = freeslot
END; ! close

GLOBAL ROUTINE initbufs =
BEGIN
INCR K FROM 0 TO nofs-1 DO
   BEGIN
   opnfl[ .K , baseadr ] = freeslot;
   opnfl[ .K , purbuf ]<purpose> = undef
   END; ! initbufs

GLOBAL ROUTINE allclose =
INCR K FROM 0 TO nofs-1 DO
   IF .opnfl[ .K , baseadr ] NEQ freeslot THEN close( .K ); ! allclose
```

```
%debug
        GLOBAL ROUTINE dmpbuf(fsp,ll,hl)=
        INCR i FROM .ll TO .hl DO
        ( ttputoct(.ttyno,.bfrl(.fsp,.i)); ttcrlf(.ttyno));

        GLOBAL ROUTINE dmptab(fsp)=
        BEGIN
        ttputoct(.ttyno,.opnf(.fsp,baseadr)); ttputoct(.ttyno,.opnf(.fsp,curadr));
        ttcrlf(.ttyno);
        ttputoct(.ttyno,.opnf(.fsp,parbuf(<purpose>));
        ttputoct(.ttyno,.opnf(.fsp,parbuf(<bufptr>)); ttcrlf(.ttyno);
        END; : dmptab
%endg%

        END
        ELUDOM
```

```
01630
01640
01650
01660
01670
01680
01690
01700
01710
01720
01730
01740
01750
01760
01770
01780
01790
01800
```

```
!*********************************************************
!****    MODULE : DISKIO
!****    DATE : 1 JULY 1982
!****    AUTHOR: S.SRINIVAS RAGHURAM
!****    ENTRIES: PUTBLK(PTRK,PSRFC,PSTR,PWRDCONT,PADDR);
!****             GETBLK(PTRK,PSRFC,PSTR,PWRDCONT,PADDR);
!****             DISKINIT();
!*********************************************************
MODULE diskio (NOLIST) =
BEGIN
MACRO
! Command register dcr
    seek     =0,1 $,      ! seek bit
    search   =1,1 $,      ! search bit
    read     =2,1 $,      ! read bit
    write    =3,1 $,      ! write bit
    restore  =4,1 $,      ! restore bit
    writechk =5,1 $,      ! write check bit
    enabl    =6,1 $,      ! interrupt enable bit
    unit     =10,1 $,     ! unit no
    port     =11,2 $,     ! 2 bit port address
    varb     =13,1 $,     ! variable / fixed format

! drive status register dsr
    sercom   =0,1 $,      ! search complete
    sekcom   =1,1 $,      ! seek complete
    oprejt   =4,1 $,      ! operation rejected
    donlin   =5,1 $,      ! drive online
    ready    =6,1 $,      ! ready to seek,read or write
    writeprot=15,1 $,     ! port write protected

! controller status register csr
    busy     =0,1 $,      ! data control busy
    bsgrtl   =1,1 $,      ! bus grant late
    crcerr   =2,1 $,      ! crcc error
    syncerr  =3,1 $,      ! synchronous error
    rwdsble  =6,1 $,      ! r/w disable
    conready =7,1 $,      ! controller ready
    dunsafe  =8,1 $,      ! drive unsafe
    cmdchk   =9,1 $,      ! command check
    sovrun   =10,1 $,     ! sector over run
    tovrun   =11,1 $,     ! track over run
    nonexmm  =12,1 $,     ! non existant memory
    wrdctofl =13,1 $,     ! word count overflow
    done     =14,1 $,     ! done flag
    error    =15,1 $,     ! error flag

    sectr    =0,5 $,      ! sector address field in str
    srface   =5,4 $,      ! surface address field in str
    bits4    =0,4 $,      ! first 4 bits
    bits5    =0,5 $;      ! first 5 bits
```

```
BIND
    str     = #177444 ;  ! disk sector and surface address register
    dsr     = #177450 ;  ! drive status register
    trk     = #177452 ;  ! disk track address register
    dcr     = #177454 ;  ! command register
    csr     = #177456 ;  ! controller status register
    adar    = #177460 ;  ! current address register
    wrd     = #177462 ;  ! word count register
    dtvt    = #200    ;  ! data transfer interrupt vector
    ndtvt   = #204    ;  ! non data transfer interrupt vector
    status  = #164000 ;  ! in command reg .sense=1; port=01
    cmdinit = #004100 ;  ! port=01; int enable=1
    wrtechk = #004140 ;  ! port=01; int en=1; writechk =1
    restre  = #004020 ;  ! port=01; restore=1
    secsize = #200    ;  ! sector size is 128 words

% result values returned by putblk/getblk
  to the monitor via FMT
  result = 0 : operation completed/word count overflow
  result = 1 : write attempted on a write protected disk
  result = 2 : seek/search error ; non data transfer error
  result = 3 : bus grant late
  result = 4 : crcc error/read or write disable
  result = 5 : synchronous error
  result = 6 : drive unsafe
  result = 7 : command check
  result = 8 : sector over run
  result = 9 : track over run
  result =10 : non existant memory
  result =11 : other errors
%

OWN   cmmdword , result , attempts ;

EXTERNAL syserr;

ROUTINE seeksearchdo ( ptrk , psrfc , pstr ) =
BEGIN
str < sectr > = .pstr < bits5 > ;
IF ( .ptrk NEQ .trk ) or ( .psrfc NEQ .str < srface > ) THEN
    BEGIN
    trk = .ptrk;
    str < srface > = .psrfc < bits4 >;
    cmmdword < seek > = 1
    END;
cmmdword < search > = 1;   ! load and go to disk controller
dcr = .cmmdword
END; ! seeksearchdo

ROUTINE commonchores ( pwrdcont , paddr ) =
BEGIN
UNTIL .csr < conready > DO ;   ! busy wait for controller to getready
DO dcr = status UNTIL .dsr < ready >;   ! busy wait for drive to getready
attempts = 3; ! 3 attempts are made to read/write
```

```
          cmdword = cmdinit ;
          wrd = .pwrdcont;
          addr = .paddr;
          cmdword<varb> = ( IF .pwrdcont GTR secsize THEN 1  ELSE 0 )
END:  . commonchores

GLOBAL ROUTINE putblk ( ptrk , psrfc , pstr , pwrdcont , paddr ) =
BEGIN
commonchores (.pwrdcont , .paddr );
cmdword<write > = 1;
dcr = status;
IF .dsr<writeprot > THEN result = 1
ELSE
  DO
    BEGIN
    seeksearchdo ( .ptrk , .psrfc , .pstr ) ;
    WAIT();
    END
  UNTIL ( .attempts LEQ 0 ) or ( .result neq 4 );
  IF ( .result NEQ 0 ) THEN syserr(.result);
  . putblk
END:

GLOBAL ROUTINE getblk ( ptrk , psrfc , pstr , pwrdcont , paddr ) =
BEGIN
commonchores (.pwrdcont , .paddr );
cmdword<read > = 1;
DO
  BEGIN
  seeksearchdo ( .ptrk , .psrfc , .pstr ) ;
  WAIT();
  END
UNTIL ( .attempts LEQ 0 ) or ( .result NEQ 4 ) ;
IF ( .result NEQ 0 ) THEN syserr(.result);
. getblk
END:

ROUTINE INTERRUPT nondata =
BEGIN
LOCAL temp ;
temp = .csr ;
IF .temp<error > THEN  result = 2
              ELSE result = 0
END:  . nondata

ROUTINE INTERRUPT data =
BEGIN
LOCAL temp;
temp = .csr;
result = ( IF .temp < error > THEN
           IF .temp< bsqrtl > THEN 3
           ELSE IF .temp< crcerr > THEN 4
           ELSE IF .temp< syncerr > THEN 5
           ELSE IF .temp< rwdsble > THEN 4
           ELSE IF .temp< dunsafe > THEN 6
           ELSE IF .temp< cmmdchk > THEN 7
```

01090
01100
01110
01120
01130
01140
01150
01160
01170
01180
01190
01200
01210
01220
01230
01240
01250
01260
01270
01280
01290
01300
01310
01320
01330
01340
01350
01360
01370
01380
01390
01400
01410
01420
01430
01440
01450
01460
01470
01480
01490
01500
01510
01520
01530
01540
01550
01560
01570
01580
01590
01600
01610
01620

```
            ELSE IF .temp < sovrun   > THEN 8
            ELSE IF .temp < tovrun   > THEN 9
            ELSE IF .temp < nonexam  > THEN 10
            ELSE IF .temp < wrdctofl > THEN 0
            ELSE 11
                    ELSE 0 ) ;
            attempts = .attempts + 1 ;
    END; ! data
END; ! data

GLOBAL ROUTINE diskinit =
    BEGIN
    dtvt = data ;  ! put address of data interrupt service routine
    ( dtvt + 2 ) = #240 ;  ! ps ; priority 5, gpr 1
    ndtvt = nondata ;
    ( ndtvt + 2 ) = #240;
    DO acr = status UNTIL .dsr < donlin > ;  ! later put a message to operator
    trk = 0; str = 0;
    acr = ; restre
    END; ! diskinit

END ELUDOM
```

016300
016400
016500
016600
016700
016800
016900
017000
017100
017200
017300
017400
017500
017600
017700
017800
017900
018000
018100
018200
018300
018400

```
****************************************************************
****************************************************************
****************************************************************
****    MODULE :: ABORTMAP                                  ****
****    DATE :: 1 JULY 1982                                 ****
****    AUTHOR :: S SRINIVAS RAGHURAM                       ****
****    INTERRUPT SERVICE ROUTINE FOR  MAPABORT             ****
****    RESTORES REGISTERS AND CALLS SWAPPOLICY             ****
****                                                        ****
****    GLOBAL ROUTINE INTERRUPT abortmap =                 ****
****       PRIORITY LEVEL = 4;                              ****
****       REGISTER SET USED IS 1                           ****
****       SUPERVISOR MODE                                  ****
****************************************************************
****************************************************************
****************************************************************

BEGIN
%.The MAPABORT interrupt service routine uses register set 1 ;
% Any program segment using register set 1 should not cause a
% Map Abort condition . If caused results are unpredictable
%

LABEL 12,15;
    INLINE(" MOV      (SP)+,@#PCOLD  ");
    INLINE(" MOV      (SP)+,@#PSOLD  ");
    ! now all registers except PC are in their old state
    ! RESTORE REGISTERS
    ! note again : ONLY REGISTER SET 0 CONSIDERED FOR RESTORATION
    %debug% dmpregs();

HALT();

12: BEGIN
    t1 = 2 * .mapsr1<8,3> + rsba;
    ! address calculation of register modified during the execution
    ! of the instruction that caused MAP ABORT
    IF .t1 EQL rsba THEN LEAVE 12;
    t2 = (IF .mapsr1 LSS 0 THEN -1 ELSE 0);
    ! the 3 bit count given by mapsr1 is in 2's complement form
    t2<0,2> = .mapsr1<13,2>; ! load count
    .t1 = .t2; ! restore register contents
    END;

15: BEGIN
    t1 = 2 * .mapsr1<0,3> + rsba;
    ! repeat for the next register
    IF .t1 EQL rsba THEN LEAVE 15;
    t2 = (IF .mapsri<0,8> LSS 0 THEN -1 ELSE 0);
    t2<0,2> = .mapsri<5,2>;
    .t1 = .t2;
    END;
    pcold = .mapsr2;

    INLINE(" MOV   @#PSOLD,-(SP) ");
    INLINE(" MOV   @#PCOLD,-(SP) ");
    %debug% dmpregs();
HALT();
```

```
00550          :    swappolicy();
00560          ;%debug% dmpcoretable();
00570          ;HALT();
00580          ;  END; ! abortmap
00590
00600          Rs0==%0
00610          Rs1==%1
00620          Rs2==%2
00630          Rs3==%3
00640          Rs4==%4
00650          Rs5==%5
00660          SP==%6
00670          PC==%7
00680
00690
00700
00710
00720  ABORTMAP:
00730          MOV     (SP)+,@#PCOLD
00740          MOV     (SP)+,@#PSOLD
00750          ;JSR    PC,DMPREGS      ;DEBUG
00760          ;HALT                   ;DEBUG
00770          MOV     #T1,Rs0
00780          MOV     @#174202,Rs1
00790          SWAB    Rs1
00800          BIC     #-7,Rs1
00810          ASL     Rs1
00820          ADD     #177740,RS1
00830          MOV     Rs1,@Rs0
00840          MOV     Rs1,#-40
00850          CMP     L2,
00860          BEQ     @#174202
00870          TST     L3,
00880          BGE     @#-1,T2
00890          MOV     L4
00900          BR      T2
00910  L3:     CLR     @#174202,Rs2
00920  L4:     MOV     Rs2
00930          ROL     Rs2
00940          ROL     Rs2
00950          ROL     Rs2
00960          BIC     #-4,Rs2
00970          BICB    #3,@#T2
00980          BISR    Rs2,@#T2
00990          SUB     @#T2,@Rs1
01000  L2:     MOVB    @#174202,Rs1
01010          BIC     #-7,Rs1
01020          ASL     Rs1,Rs1
01030          ADD     #177740,RS1
01040          MOV     Rs1,@Rs0
01050          CMP     Rs1,#-40
01060          BEQ     L5
01070          TSTB    @#174202
01080          BHIS    L6
```

```
001090          MOV    #-1,T2
001100          BR     L7
001110   L6:    CLR    T2
001120   L7:    MOVR   @#174202,RS2
001130          ASL    RS2
001140          ASL    RS2
001150          SWAB   RS2
001160          BIC    #-4,RS2
001170          BICB   #3,@#T2
001180          BISR   RS2,@#T2
001190          SUB    @#T2,@RS1
001200          MOV    @#174204,PCOLD
001210          MOV    @#PSOLD,-(SP)
001220   L5:    MOV    @#PCOLD,-(SP)
001230          ;JSR   PC,DMPREGS      ;DEBUG
001240          HALT                   ;DEBUG
001250          JSR    PC,SWAPPOLICY
001260          JSR    PC,DMPCORETAB
001270   ::     HALT                   ;DEBUG
001280          RTI                    ;DEBUG
001290
001300
001310          .CSECT ABRT.0
001320   T1:    .=.+2
001330   T2:    .=.+2
001340   PCOLD: .=.+2
001350   PSOLD: .=.+2
001360
001370          .GLOBL ABORTMAP
001380          .GLOBL SWAPPOLICY
001390          ..GLOBL DMPREGS        ;DEBUG
001400          .GLOBL DMPCORETAB      ;DEBUG
001410
001420          .END
001430
```

```
%***********************************************************************
************  MODULE :  MEMORYMANAGER
************  DATE : 1 JULY 1987
************  AUTHOR :  S SRINIVAS RAGHURAM
************  ENTRIES:  SWAPPOLICY(); INITMAP();
************
***********  MODULE memorymanager ( list ) =

BEGIN
MACRO   ! subfields in segdes field of coretable
  segtyp = 3,2 $ ,
  segno  = 0,3 $ ,

  %      ! fields in segmenttable descriptor register
  % = 6,1 $ ;  ! segment written into

MACRO
incparcelholder = .parcelholder + 1;
  parcelholder = .parcelholder + 1;
  IF .parcelholder GTR coretabmax THEN
  parcelholder = 0 $;

MACRO
passparcel =
DO ( incparcelholder)
  WHILE .ctl.parcelholder,resident] $;

BIND   ! of hwsegment table
  mdr = 0 , ,   !map descriptor register
  mar = 1 , ,   !map address register

  ! of core table
  segdes   = 0 , ,   ! segment descriptor
  resident = 1 ,     ! permanently resident segment

  ! registers / addresses
  mapsr0   = #174200 , ,   map status register 0
  mapsr1   = #174202 , ,   map status register 1
  mapsr2   = #174204 , ,   map status register 2
  rsbase   = #177740 , ,   register set base address
  mapabt   = #54 , ,       MAPABORT interrupt vector
  maptrp   = #44 , ,       MAPTRAP interrupt vector
  outcore  = #077400 ,

  ! for swapping
  onek    = 1024 , ,   page size in words
  twok    = 2048 , ,   page size in bytes
  segsize = 8192 ;   ! segment size in bytes
```

```
! constants
du        = 0      !.user data space
ds        = 1      !.supervisor data space
iu        = 2      !.user instruction space
is        = 3      !.supervisor instruction space
free      = #40    !.free space
coretabmax = 7
segtabmax  = 31
false      = 0
true       = 1

STRUCTURE nwsegmenttable [ i1 , i2 , j ] =
CASE .j OF
SET
  ( .nwsegmenttable + .i1 * 32 + 2 * .i2 );
  ( .nwsegmenttable + .i1 * 32 + 2 * .i2 + 16 )
TES;

STRUCTURE coretable [ i , j ] =
CASE .j OF
SET
  ( .coretable + .i * 4 + 2 );
  ( .coretable + .i * 4 + 2 );
TES;

STRUCTURE swsegmenttable [ i , j ] =
( .swsegmenttable + 2 * .i );

EXTERNAL putblk, getblk;
%debug% external putlch,ttputoct,ttcrlf,ttyno;

OWN coretable ctl (coretabmax + 1) * 2 ];

BIND nwsegmenttable st = #174000;

OWN swsegmenttable swst [ segtabmax + 1 ];

EXTERNAL abortmap;

OWN parcelholder;

ROUTINE loadseg ( virtsegno , physegno ) =
BEGIN
LOCAL strtadr;
strtadr = .physegno * segsize;
stf[ .virtsegno<segtyp> , .virtsegno<segno> , nar ] = .strtadr DIV 64 ;
stf[ .virtsegno<segtyp> , .virtsegno<segno> , ndr ] = .swstf .virtsegno ] ;
ctf[ .physegno , segdes ] = .virtsegno;
ctf[ .physegno , resident ] = false;
INCR srfc FROM 0 TO 3 DO
  BEGIN
    getblk( .virtsegno , srfc , 2 , onek , .strtadr );
    strtadr = .strtadr + twok
```

```
005500
005600
005700
005800
005900
006000
006100
006200
006300
006400
006500
006600
006700
006800
006900
007000
007100
007200
007300
007400
007500
007600
007700
007800
007900
008000
008100
008200
008300
008400
008500
008600
008700
008800
008900
009000
009100
009200
009300
009400
009500
009600
009700
009800
009900
010000
010100
010200
010300
010400
010500
010600
010700
010800
```

80

```
        END;
END; ! loadseg
ROUTINE storeseg ( virtsegno , physegno ) =
BEGIN
    LOCAL strtadr;
    strtadr = .physegno * segsize;
    stf[ .virtsegno<segtyp> ] .virtsegno<segno> ,mdr ] = 0;
    ctl[ .physegno , segdes ] = free;
    INCR srfc FROM 0 TO 3 DO
    BEGIN
        putblk( .virtsegno , .srfc , 2 , onek , .strtadr );
        strtadr = .strtadr + twok
    END;
END; ! storeseg

GLOBAL ROUTINE swappolicy =
BEGIN
    LOCAL k; ! find the segment that caused the fault
    K<3,1> = 0;                 ! U/S mode
    K<4,1> = NOT .mapsr0<6,1>;  ! U/S mode
    K<5,1> = NOT .mapsr0<5,1>;  ! D/I space
    K<0,3> = .mapsr0<2,3>;      ! segment number
    IF .ctl[ .parcelholder , segdes ] EQL .K THEN
    BEGIN
        LOCAL K1;
        stl[ .K<segtyp> ,.K<segno> ,mdr] = .swstl .K ];
        passparcel;
        K1 = .ctl[ .parcelholder , segdes ];
        stl[ .K1<segtyp> , .K1<segno> ,mdr ] = outcore.
    END
    ELSE
    BEGIN
        BIND K1 = ctl[ .parcelholder , segdes ];
        IF .stl[ .K1<segtyp>,.K1<segno> , mdr ]<w> THEN
            storeseg(.K1,.parcelholder)
        ELSE
            stl[ .K1<segtyp> ,.K1<segno> , mdr ] = 0;
        loadseg( .K, .parcelholder );
        passparcel;
    END;
    mapsr0<12,4> = 0 ; ! reset bits to resume MAP functioning from next instruction
END; ! swappolicy
%debug%
GLOBAL ROUTINE dmpcoretable=
BEGIN
    LOCAL n;
    INCR i FROM 0 TO coretabmax DU
    BEGIN
        n = .ctl[ .i , segdes ]<segtyp>;
        ttputoct(.ttyno,.n); putlch(.ttyno,#40);
        n = .ctl[ .i , segdes ]<segno>; putlch(.ttyno,#40);
        ttputoct(.ttyno,.n); putlch(.ttyno,#40);
```

```
            n = .ctl[.i,resident];
            ttputoct(.ttvno,.n); ttcrlf(.ttvno)
            END;
        ttputoct(.ttvno,parcelholder)
        END; ! dmpcoretable

GLOBAL ROUTINE dmpregs=
    BEGIN
    BIND strtadr = #177744;
    INCR i FROM 0 TO 6 BY 2 DO
        BEGIN
        ttputoct(.ttvno,.(strtadr+.i)); putlch(.ttvno,#40)
        END;
    ttcrlf(.ttvno)
    END; ! dmpregs
%debug

GLOBAL ROUTINE initmap =
    BEGIN
    INCR i FROM 0 TO segtabmax DO
        BEGIN
        swstl[.i,j] = #077407; !maplf==#177 ; ACF==#7
        stl[.i<segtyp> , .i<seqno> , mdr ] = #077400; !maplf==#177 ; ACF==#0
        END;
    INCR i FROM 0 TO coretabmax DO
        BEGIN
        ctl[.i , seqdes ] = tree; !free
        ctl[.i , resident ] = false;
        END;
        stl[ is , mar ] = 0; !initialize seg0 to present code
        stl[ is , mar ] = #7600; !registers
        stl[ is , mdr ] = #077407; ! maplf==#177 , acf = #7
        stl[ ds , mdr ] = #7600; stl[ ds , 7 , mdr ] = #077407;
        stl[ iu , mdr ] = #7600; stl[ iu , 7 , mdr ] = #077407;
        stl[ du , mdr ] = #7600; stl[ du , 7 , mdr ] = #077407;
        ctl[ 00 , seqdes ] = #30; !is,seg0
        ctl[ 00 , resident ] = true; !free
        ctl[ 7 , seqdes ] = #37; !is,seg7
        ctl[ 7 , resident ] = true; !is,seg7
        ctl[ 6 , resident ] = true; !debug
        ctl[ 5 , resident ] = true; !debug
        ctl[ 4 , resident ] = true; !debug
        ctl[ 3 , resident ] = true; !debug
        parcelholder = -1; passparcel;
        mapapt = aportmap;
        (mapabt + 2) = #1200; ! priority 4 ; reg-set 1
        mapsr0 = #000001; ! enable mapping - bit 0
    END; ! initmap
    END
ELUDOM
```

001630
001640
001650
001660
001670
001680
001690
001700
001710
001720
001730
001740
001750
001760
001770
001780
001790
001800
001810
001820
001830
001840
001850
001860
001870
001880
001890
001900
001910
001920
001930
001940
001950
001960
001970
001980
001990
002000
002010
002020
002030
002040
002050
002060
002070
002080
002090
002100
002110
002120
002130
002140

```
;*********************************************...*********
;*****                                              *****
;*****      MODULE : ACCESSTUPCODE                  *****
;*****                                              *****
;*********************************************...*********
MODULE commnlib( nolist ) =
BEGIN
BYTE OWN  VECTOR pcode(#200001);

    GLOBAL ROUTINE deposit(adr,value) =
        pcode(.adr) = .value;

    GLOBAL ROUTINE fetch(adr) =
        .pcode(.adr);

END
ELUDOM

;MODULE commnlib( nolist ) =
; BEGIN
;  GLOBAL ROUTINE deposit(adr,value) =
;    .adr = .value;
;
;  GLOBAL ROUTINE fetch(adr) =
;    return ..adr;
;end
;eludom

    .PDP10
    .TITLE  COMMNLIB
    .CSECT  COMM.C

RS0==0
RS1==1
RS2==2
RS3==3
RS4==4
RS5==5
SP==6
PC==7

DEPOSIT: ;DEPOSIT(ADR,VALUE)=
    .WORD   22              ;TKM 2 , TO UI
    MOV     2(SP),@4(SP)    ;.ADR = .VALUE
    .WORD   20              ;TKM 0 , .NORMAL
    RTS     PC
```

```
FETCH:   ;FETCH(ADR)=

         .WORD    23          ;TKM 3   , FROM III
         MOV      @2(SP),RS0  ;RETURN  ,.ADR
         .WORD    20          ;TKM 0 , .NORMAL
         RTS      PC


COMMNLIB:

         .GLOBL  COMMNLIB
         .GLOBL  FETCH
         .GLOBL  DEPOSIT


         .END
```

```
00550
00560
00570
00580
00590
00600
00610
00620
00630
00640
00650
00660
00670
00680
00690
00700
00710
00720
00730
00740
00750
00760
00770
00780
00790
00800
00810
00820
00830
00840
00850
```

```
%*********
**********        MODULE : ARITHMETICLIBRARY
**********
******* *    .CSECT ARITH.C
********     .TITLE ARITH.C
********
********     MO=177736
********     Rs0=%0
********     Rs1=%1
********     Rs2=%2
********     Rs3=%3
********     Rs4=%4
********     Rs5=%5
********     SP=%6
********     PC=%7

MUL:   MOV  2(SP),@#MO
       .WORD 163121
       TST  (SP)+              ;MUL 4(SP),-(SP)
       MOV  @#MQ,Rs0
       RTS  PC

MODR:  MOV  4(SP),@#MO
       CLR  Rs0
       .WORD 113107
       .WORD 2                 ;DIV 2(SP),Rs0
       RTS  PC

DIVR:  MOV  4(SP),@#MO
       CLR  Rs0
       .WORD 113107
       .WORD 2                 ;DIV 2(SP),Rs0
       MOV  @#MO,Rs0
       RTS  PC

SHIFT: JSR  Rs1,sSAV2
       MOV  #12,Rs0
       ADD  SP,Rs0
       MOV  10(SP),Rs2
       BGE  Ls5
       NEG  10(SP)
       MOV  #1,Rs1
       BR   Ls7
Ls6:   ASR  @Rs0
       INC  Rs1
Ls7:   CMP  Rs1,10(SP)
       BLE  Ls6
       BR   Us2
Ls5:   MOV  #1,Rs1
       BR   Ls10
```

```
000010
000020
000030
000040
000050
000060
000070
000080
000090
000100
000110
000120
000130
000140
000150
000160
000170
000180
000190
000200
000210
000220
000230
000240
000250
000260
000270
000280
000290
000300
000310
000320
000330
000340
000350
000360
000370
000380
000390
000400
000410
000420
000430
000440
000450
000460
000470
000480
000490
000500
000510
000520
000530
000540
```

```
00550   LS9:    ASL    @RS0
00560   LS10:   INC    RS1
00570           CMP    RS1,RS2
00580           BLE    LS9
00590   US2:    MOV    @RS0,RS0
00600           RTS    PC
00610
00620   ; ROUTINE SIZE:  28
00630
00640           .GLOBL  MUL
00650           .GLOBL  DIVR
00660           .GLOBL  MODR
00670           .GLOBL  SHIFT
00680           .GLOBL  $SAV2
00690           .END
```

APPENDIX - 7
****

!DOCUMENTATION OF CHANGES DONE TO THE P-COMPILER
!**

The present version of the P-Compiler called
PAS316.PAS is based on the original Pascal-P Compiler
developed at ETH Zurich and described in [ ] Compiler

The Compiler had been modified to enable efficient
interpretation of the P-Code it generates on TDC-316.
Major modifications to the P-Compiler are as under:

a) CONST
   intsize = 2; realsize = 4; charsize = 2; boolsize = 2;
   setsize = 8; ptrsize = 2; lcaftermarkstack = 10;

b) VAR
   ppc , maxppc : integer;
   (* P-Machine Program Counter. This counter keeps the count
      of the number of bytes of code generated, as each P-Code
      instruction is emitted by the Compiler. It takes into
      account the assembler generated optimizations also. This
      counter's value is displayed at the beginning of each
      line of the listing produced by the compiler, and thus
      enables the interpreter to point a run-time error correct
      to one source line of the user program. *)

   runtmchk : boolean;
   (* the Compiler can now optionally generate code for check-
      ing of array index bounds and subrange bounds at run-
      time. Default : ON *)

   errflag : boolean;
   (* this flag gets set when an error is detected by the Com-
      piler. *)

c) PROCEDURE
   i) insymbol;
      (* insymbol has now been split up into three smaller rout-
      a) inident;
         (* accepts lower case alphabets and underscore character *)
      b) innumber;
      c) insymbol;
         (* insymbol now accounts for all printable characters *)

   ii) getadch;
      (* this routine is newly introduced. getadch is invoked by
         the compiler during code generation. It checks the attr-
         ibutes of the expression being parsed, and returns the
         type information in the variable ADCH. *)

   iii) gen0 ,gen1 ,gen2 ,load, store, loadaddress;

```
     (* these procedures have been modified to generate the
        appropriate type information with the p-code mnemonics
        and write the correct PPC value.                     *)

iv)  selector;
     (* array index bounds checks introduced               *)
v)   assignment;
     (* subrange bounds checks introduced                  *)
vi)  casestatement;
     (* the code generated is now CHKS LB UB for range
        checking and the UJC for undefined case labels      ;*)
     see Fig 2.1
vii) (* All places where load and store instructions are gen-
        erated, now if feasible generate the newly introduced
        instructions LDL ,STL and LAL.                      *)
```

APPENDIX - R
****

!PAS316 USER MANUAL
****

A user can interact with the PAS316 System through the
Command Processor. The Command Processor accepts commands
from the TTY, and initiates the various routines necessary
to execute the Command.

A set of switches have been provided to enable the user
to select and deselect some facilities of the PAS316 System.
Each occurrence of the switch complements the current status
of the switch.
   for example:
      if switch 0 was off, then
      /O/O   -- keeps it on.
      /O/O/O   -- switches it off.

SWITCHES:
****
   L: Listing of the Assembly Language Program, by the Assembler.
   O: Display the contents of the file OUTPUT, as they are writ-
      ten by the P-Interpreter on the TTY. This feature is useful
      for observing the results of a program execution as they
      are obtained. Especially meant to observe the listfile
      as it is generated by the compiler on the TTY.
   S: Save the Object Code produced by the P-Assembler in a file.
   T: Produce the Execution Trace of the P-Instruction execution.
      Displays on the TTY after every instruction execution the
      values of the following registers:
      Opcode, Program Counter, and the contents of the top 4
      words of the Stack.

      There are a set of Commands to facilitate the user to run
his programs. The Commands ask for filenames where needed.

COMMANDS:
****

   COPY   : Copies one text file into another.
            Switches : NONE

   TYPE   : Types a file on the TTY.
            Switches : NONE

   PASCAL : Compiles a Pascal program. Files used are:
            INPUT - must contain the source program.
            OUTPUT- optionally a source listing is produced,
                    and Compiler messages.

PKR = optionally the P-Code is produced.
Switches : NONE

PASASM : Compiles the source program and assembles the
P-Assembly Language produced by it.
Switches : L: default = OFF
S: default = ON

PASGO : Compiles and then assembles and executes the
Pascal program.
Switches : L: default = OFF
O: default = ON
S: default = OFF
T: default = OFF

ASM : Assembles the P-Machine's Assembly Language program.
Switches : L: default = OFF
S: default = ON

ASMINT : Assembles and interprets the P-Machine's Assembly
Language program.
Switches : L: default = OFF
O: default = ON
S: default = OFF
T: default = OFF

EX : Loads and interprets the given P-Code.
Switches : O: default = ON
T: default = OFF

KJOB : Exit from PAS316.